普通高等教育“十一五”国家级规划教材

普通高等教育精品教材

新世纪计算机基础教育丛书 | 丛书主编 谭浩强

计算机软件技术基础

（第四版）

徐士良 葛兵 编著

清华大学出版社

VISIT…

LANZAROTE Caliente.com

有关此电子书的说明

本人可以帮助你找到你要的PDF电子书，计算机类，文学，艺术，设计，医学，理学，经济，金融等等。质量都很清晰，为方便读者阅读观看，每本100%都带可跳转的书签索引和目录，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 QQ1779903665。

PDF代找说明：
本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ，大部分我都可以找到，而且每本100%带书签索引目录。因PDF电子书都有版权，请不要随意传播，如果您有经济购买能力，请尽量购买正版。

提供各种书籍的pd电子版代找服务，如果你找不到自己想要的书的pdf电子版，我们可以帮您找到，如有需要，请联系 QQ 1779903665.

备用:QQ 461573687
若以上联系方式失效，您可通过以下电子邮件获取有效联系方式。
E-mail：ebooksprite@163.com
E-mail：ebooksprite@foxmail.com

若您没有QQ通讯工具，请发送您的请求到 ebooksprite@gmail.com 与客服取得联系。

声明：本人只提供代找服务，每本100%索引书签和目录，因寻找和后期制作pdf电子书有一定难度，仅收取代找费用。如因PDF产生的版权纠纷，与本人无关，我们仅仅只是帮助你寻找到你要的pdf而已。

内容简介

- 系统性。以数据结构与算法为基础，以软件技术为线索，介绍软件开发中所需要的软件知识和技术。
- 应用性。内容围绕解决软件开发中所遇到的软件技术问题展开。在介绍系统软件（如操作系统）时，为了便于读者理解，也适当介绍一些原理，但主要还是介绍实现系统软件中的技术，以便读者将这些技术用到应用软件的开发中。
- 可读性。深入浅出，通过实例引出基本概念，便于教学。

清华大学出版社数字出版网站
WQBook 文泉书局
www.wqbook.com

ISBN 978-7-302-35719-3
9 787302 357193 >
定价：39.50元

普通高等教育"十一五"国家级

普通高等教育精品教材

新世纪计算机基础教育丛书 | 丛书主编 谭浩强

计算机软件技术基础

（第四版）

徐士良　葛兵　编著

清华大学出版社

北 京

内 容 简 介

本书针对高等学校的大学生、研究生以及科技工作者与研究人员学习计算机软件应用技术的需要，介绍了计算机软件设计的基础知识、方法与实用技术。书中主要内容包括集合与算法的基本概念、基本数据结构及其运算、查找与排序技术、资源管理技术、数据库设计技术、编译技术概述、应用软件设计与开发技术。每章都配有一定数量的习题。

本书内容丰富、通俗易懂、实用性强，可作为高等学校的教材，也可作为广大从事计算机应用工作的科技人员的参考书。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目(CIP)数据

计算机软件技术基础/徐士良，葛兵编著．--4版．--北京：清华大学出版社，2014
新世纪计算机基础教育丛书
ISBN 978-7-302-35719-3

Ⅰ．①计…　Ⅱ．①徐…②葛…　Ⅲ．①软件—技术　Ⅳ．①TP31

中国版本图书馆CIP数据核字(2014)第060808号

责任编辑：焦　虹
封面设计：傅瑞学
责任校对：时翠兰
责任印制：刘海龙

出版发行：清华大学出版社
　网　　址：http://www.tup.com.cn，http://www.wqbook.com
　地　　址：北京清华大学学研大厦A座　　邮　编：100084
　社 总 机：010-62770175　　邮　购：010-62786544
　投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
　质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
　课 件 下 载：http://www.tup.com.cn，010-62795954
印 装 者：三河市中晟雅豪印务有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　印　张：23.5　　字　数：563千字
版　　次：2002年2月第1版　2014年9月第4版　　印　次：2014年9月第1次印刷
印　　数：1～2000
定　　价：39.50元

产品编号：053692-01

丛书序言

现代科学技术的飞速发展，改变了世界，也改变了人类的生活。作为新世纪的大学生，应当站在时代发展的前列，掌握现代科学技术知识，调整自己的知识结构和能力结构，以适应社会发展的要求。新世纪需要具有丰富的现代科学知识，能够独立完成面临的任务，充满活力，有创新意识的新型人才。

掌握计算机知识和应用，无疑是培养新型人才的一个重要环节。现在计算机技术已深入到人类生活的各个角落，与其他学科紧密结合，成为推动各学科飞速发展的有力的催化剂。无论学什么专业的学生，都必须具备计算机的基础知识和应用能力。计算机既是现代科学技术的结晶，又是大众化的工具。学习计算机知识，不仅能够掌握有关知识，而且能培养人们的信息素养。这是高等学校全面素质教育中极为重要的一部分。

高校计算机基础教育应当遵循的理念是：面向应用需要，采用多种模式，启发自主学习，重视实践训练，加强创新意识，树立团队精神，培养信息素养。

计算机应用人才队伍由两部分人组成：一部分是计算机专业出身的计算机专业人才，他们是计算机应用人才队伍中的骨干力量；另一部分是各行各业中应用计算机的人员。这后一部分人一般并非计算机专业毕业。他们人数众多，既熟悉自己所从事的专业，又掌握计算机的应用知识，善于用计算机作为工具解决本领域中的任务。他们是计算机应用人才队伍中的基本力量。事实上，大部分应用软件都是由非计算机专业出身的计算机应用人员研制的，他们具有的这个优势是其他人难以代替的。从这个事实可以看到在非计算机专业中深入进行计算机教育的必要性。

非计算机专业中的计算机教育，无论目的、内容、教学体系、教材、教学方法等各方面都与计算机专业有很大不同，绝不能照搬计算机专业的模式和做法。全国高等院校计算机基础教育研究会自 1984 年成立以来，始终不渝地探索高校计算机基础教育的特点和规律。2004 年，全国高等院校计算机基础教育研究会与清华大学出版社共同推出了《中国高等院校计算机基础教育课程体系 2004》（简称 CFC2004）。2006 年、2008 年又分别推出了《中国高等院校计算机基础教育课程体系 2006》（简称 CFC2006）及《中国高等院校计算机基础教育课程体系 2008》（简称 CFC2008），由清华大学出版社正式出版发行。

1988 年起，我们根据教学实际的需要，组织编写了《计算机基础教育丛书》，邀请有丰富教学经验的专家、学者先后编写了多种教材，由清华大学出版社出版。丛书出版后，迅速受到广大高校师生的欢迎，对高等学校的计算机基础教育起了积极的推动作用。广大读者反映这套教材定位准确，内容丰富，通俗易懂，符合大学生的特点。

1999 年，根据新世纪的需要，在原有基础上组织出版了《新世纪计算机基础教育丛书》。由于丛书内容符合需要，质量较高，因此被许多高校选为教材。丛书总发行量 1000 多万册，这在国内是罕见的。最近，我们又对丛书进行了进一步的修订，根据发展的需要，增加了许多新的教材。本丛书有以下特点：

（1）内容新颖。根据 21 世纪的需要，重新确定丛书的内容，以符合计算机科学技术的发展和教学改革的要求。本丛书除保留了原丛书中经过实践考验且深受群众欢迎的优秀教材外，还编写了许多新的教材。在这些教材中反映了近年来迅速得到推广应用的一些计算机新技术，以后还将根据发展不断补充新的内容。

（2）满足不同学校的教学需求。本丛书采用模块形式，提供了各种课程的教材，内容覆盖了高校计算机基础教育的各个方面。丛书中既有理工类专业的教材，也有文科和经济类专业的教材；既有必修课的教材，也包括一些选修课的教材。各类学校都可以从中选择到合适的教材。

（3）符合初学者的特点。本丛书针对初学者的特点，以应用为目的，以应用为出发点，强调实用性。本丛书的作者都是长期在第一线从事高校计算机基础教育的教师，对学生的基础、特点和认识规律有深入的研究，在教学实践中积累了丰富的经验。可以说，每一本教材都是他们长期教学经验的总结。在教材的写法上，作者既注意概念的严谨和清晰，又特别注意采用读者容易理解的方法阐明看似深奥难懂的问题，力求做到例题丰富，通俗易懂，便于自学。这一点是本丛书一个十分重要的特点。

（4）采用多样化的形式。除了教材这一基本形式外，有些教材还配有习题解答和实验指导，并提供电子教案。

总之，本丛书的指导思想是内容新颖、概念清晰、实用性强、通俗易懂、教材配套，简单概括为："新颖、清晰、实用、通俗、配套"。我们经过多年实践形成的这一套行之有效的创作风格，相信会受到广大读者的欢迎。

本丛书多年来得到了各方面人士的指导、支持和帮助，尤其是得到了全国高等院校计算机基础教育研究会的各位专家和高校老师的支持和帮助，我们在此表示由衷的感谢。本丛书肯定有不足之处，希望得到广大读者的批评指正。

欢迎访问谭浩强网站：http：//www.tanhaoqiang.com

丛书主编
全国高等院校计算机基础教育研究会荣誉会长
谭浩强

第四版 前言

本书第二版是普通高等教育“十一五”国家级规划教材，并被评为2008年度普通高等教育精品教材。

本次修订保持了前三版的特点，在第三版的基础上进行了如下几方面的调整与修改。

（1）所有算法程序均采用C++语言进行描述。由于C++语言既可用于面向过程的程序设计，又支持面向对象的程序设计，因此，作者在对算法进行描述时，尽量采用最合适的程序设计方法。例如，对于同一批数据进行同类操作的单个算法采用面向过程的方法；对于基本的数据结构（如顺序存储与链式存储的线性表、栈、队列等）采用面向对象的方法，将数据与运算封装成类，以便在其他应用程序中直接使用。这样，书中的所有算法都可以直接在实际应用中方便地使用。

（2）在第2章的2.7节中增加了最短距离问题的内容。

（3）在第3章中增加了字符串匹配的内容。

本书内容丰富、通俗易懂、实用性强，书中所有算法程序均上机调试通过。本书可作为高等学校的大学生或研究生软件课程的教材，也可作为广大从事计算机应用工作的科技人员的参考书。

由于作者水平有限，书中难免有错误或不妥之处，恳请读者批评指正。

作　者

第三版 前　言

本书第二版是普通高等教育"十一五"国家级规划教材，并被评为2008年度普通高等教育精品教材。

本次修订保持了第二版的特点（见第二版前言），主要在内容上作了如下几方面的调整。

（1）在第1章中增加了集合方面的基本知识，对算法方面的基本内容进行了适当的精简。

（2）在第2章中增加了索引存储结构的内容。

（3）在第5章中删去了关系数据库语言SQL的内容。

（4）新增加了编译技术概述一章。

本书内容丰富、通俗易懂、实用性强，书中所有算法程序均上机调试通过。本书可作为大学生或研究生的软件技术基础课程的教材，也可作为广大从事计算机应用工作的科技人员的参考书。

由于作者水平有限，书中难免有错误或不妥之处，恳请读者批评指正。

作　者

第二版 前言

高等学校非计算机专业的学生(包括广大科技人员)如何学习和掌握软件技术,是一个很重要的问题。他们不可能像计算机专业的学生那样学习软件的各门课程,因此有必要将主要的软件技术和知识在一门课程中介绍,但又不能是拼盘式的组合。国外根本没有这样的教材,国内这样的教材也不多。有的教材虽然名称叫"软件技术",但实际上是一些新软件的使用。本教材介绍的是软件技术,而不是软件的使用。

本教材主要针对高等学校非计算机专业的学生学习计算机软件技术的需要,介绍有关软件基础知识及应用技术。其特点如下:

(1) 系统性。本教材不是将计算机专业各门课程的内容简单地拼装在一起,而是针对学生对软件应用技术的需要,将其有机结合,为读者提供软件开发中所需要的软件知识和技术。全书以数据结构与算法为基础,以软件技术为线索,系统性较强。

(2) 强调应用。本教材强调以应用为目的,书中实例比较丰富,内容围绕解决软件开发中所遇到的软件技术问题来展开。在介绍系统软件(如操作系统)时,为了便于读者理解,也适当介绍一些原理,但主要还是介绍实现系统软件中的技术,以便读者将这些技术用到应用软件的开发中。

(3) 可读性强。本书深入浅出,通过实例引出基本概念,便于读者接受。

此次对本书的修订主要是前三章。书中所有的算法均采用 C++ 描述。由于 C++ 语言既可用于面向过程的程序设计,又支持面向对象的程序设计,因此,作者在对算法进行描述时,尽量采用最合适的程序设计方法。例如,对于基本的数据结构(如顺序存储与链式存储的线性表、栈、队列等)采用面向对象的方法,将数据与运算封装成类,以便在其他应用程序中直接使用;而对于同一批数据进行同类操作的各种算法(如对线性表的各种排序方法)则采用面向过程的方法,将各种不同的算法用普通函数来描述。这样,书中的所有算法都可以直接应用。

本书内容丰富、通俗易懂、实用性强,书中所有算法程序(C++ 描述)均上机调试通过。本书可作为非计算机专业的大学生或研究生软件课程的教材,也可作为广大从事计算机应用工作的科技人员的参考书。

由于作者水平有限,书中难免有错误或不妥之处,恳请读者批评指正。

作　者

第一版 前言

随着计算机技术的深入发展，计算机技术的应用已经渗透到各个领域，特别是计算机软件的设计与开发，已经不只是计算机专业人员的事情了。现在，越来越多的软件需要非计算机专业的人员来设计与开发，很多系统软件与应用软件由非计算机专业人员来使用，并在此基础上进行二次开发。因此，普及计算机软件技术已经是大势所趋。

本书在《软件应用技术基础》（该书由清华大学出版社出版，获电子工业部优秀教材一等奖）一书的基础上改写而成，可满足广大非计算机专业的学生学习软件设计与开发的需要。作为应用计算机的科技人员，除了要掌握现有计算机软件的使用外，从实际应用出发，还必须要掌握软件设计与开发的基本知识和有关技术，如数据的组织、程序的组织、计算机资源的利用、数据的处理技术等，以便得心应手地进行应用软件的设计与开发。

全书共分 6 章，每章后面都附有一定数量的习题。

第 1 章介绍算法。内容主要包括算法的基本概念、算法的基本设计方法、算法的复杂度分析等内容。

第 2 章介绍基本数据结构及其运算。内容主要包括数据结构的基本概念，线性表、栈、队列及其在顺序存储结构下的运算和应用，线性链表及其运算，数组，二叉树的概念、存储及其应用，图的存储及其遍历。

第 3 章介绍常用的查找与排序技术。内容主要包括基本的查找技术、哈希表技术、基本的排序技术、二叉排序树及其查找、多层索引树及其查找、拓扑分类。

第 4 章介绍资源管理技术。内容主要包括操作系统的功能与任务、多道程序设计、存储空间的组织。

第 5 章介绍数据库技术。内容主要包括数据库基本概念、关系代数、数据库设计、关系数据库语言 SQL。

第 6 章介绍应用软件设计与开发技术。内容主要包括软件工程概述、软件详细设计的表达、结构化分析与设计方法、测试与调试基本技术、软件开发新技术。

本书内容丰富、通俗易懂、实用性强，书中所有算法程序均上机调试通过。本书可作为高等学校非计算机专业的大学生或研究生软件课程的教材，也可作为广大从事计算机应用工作的科技人员的参考书。

由于作者水平有限，书中难免有错误或不妥之处，恳请读者批评指正。

作　者

目　录

1 预备知识

1.1　集合…… 1
　1.1.1　集合及其基本运算 …… 1
　1.1.2　自然数集与数学归纳法 …… 3
　1.1.3　笛卡儿积 …… 5
　1.1.4　二元关系 …… 5
1.2　算法…… 6
　1.2.1　算法的基本概念 …… 6
　1.2.2　算法设计基本方法 …… 8
　1.2.3　算法的复杂度分析…… 13
习题…… 15

2 基本数据结构及其运算

2.1　数据结构的基本概念 …… 17
　2.1.1　什么是数据结构…… 17
　2.1.2　数据结构的图形表示…… 20
2.2　线性表及其顺序存储结构 …… 22
　2.2.1　线性表及其运算…… 22
　2.2.2　栈及其应用…… 31
　2.2.3　队列及其应用…… 41
2.3　线性链表 …… 52
　2.3.1　线性链表的基本概念…… 52
　2.3.2　线性链表的插入与删除…… 56
　2.3.3　带链的栈与队列…… 61
　2.3.4　循环链表…… 68
　2.3.5　多项式的表示与运算…… 71
2.4　线性表的索引存储结构 …… 78
　2.4.1　索引存储的概念…… 78

2.4.2 “顺序-索引-顺序”存储方式 …… 80
2.4.3 “顺序-索引-链接”存储方式 …… 81
2.4.4 多重索引存储结构 …… 82
2.5 数组 …… 83
2.5.1 数组的顺序存储结构 …… 83
2.5.2 规则矩阵的压缩 …… 83
2.5.3 一般稀疏矩阵的表示 …… 86
2.6 树与二叉树 …… 112
2.6.1 树的基本概念 …… 112
2.6.2 二叉树及其基本性质 …… 115
2.6.3 二叉树的遍历 …… 118
2.6.4 二叉树的存储结构 …… 119
2.6.5 穿线二叉树 …… 124
2.6.6 表达式的线性化 …… 136
2.7 图 …… 138
2.7.1 图的基本概念 …… 138
2.7.2 图的存储结构 …… 139
2.7.3 图的遍历 …… 143
2.7.4 最短距离问题 …… 144
2.7.5 图邻接表类 …… 146
习题 …… 155

3 查找与排序技术

3.1 基本的查找技术 …… 158
3.1.1 顺序查找 …… 158
3.1.2 有序表的对分查找 …… 158
3.1.3 分块查找 …… 163
3.2 哈希表技术 …… 164
3.2.1 哈希表的基本概念 …… 164
3.2.2 几种常用的哈希表 …… 167
3.3 基本的排序技术 …… 185
3.3.1 冒泡排序与快速排序 …… 185
3.3.2 简单插入排序与希尔排序 …… 190
3.3.3 简单选择排序与堆排序 …… 192
3.3.4 其他排序方法简介 …… 195
3.4 二叉排序树及其查找 …… 199

3.4.1 二叉排序树的基本概念 …… 200
3.4.2 二叉排序树的插入 …… 201
3.4.3 二叉排序树的删除 …… 202
3.4.4 二叉排序树查找 …… 204
3.5 多层索引树及其查找 …… 206
3.5.1 B^- 树 …… 207
3.5.2 B^+ 树 …… 217
3.6 拓扑分类 …… 227
3.7 字符串匹配 …… 229
3.7.1 字符串的基本概念 …… 229
3.7.2 字符串匹配的 KMP 算法 …… 230
习题 …… 235

4 资源管理技术

4.1 操作系统的概念 …… 237
4.1.1 操作系统的功能与任务 …… 237
4.1.2 操作系统的发展过程 …… 238
4.1.3 操作系统的分类 …… 241
4.2 多道程序设计 …… 244
4.2.1 并发程序设计 …… 244
4.2.2 进程 …… 247
4.2.3 进程之间的通信 …… 251
4.2.4 多道程序的组织 …… 256
4.3 存储空间的组织 …… 257
4.3.1 内存储器的管理技术 …… 257
4.3.2 外存储器中文件的组织结构 …… 261
习题 …… 268

5 数据库设计技术

5.1 数据库基本概念 …… 269
5.1.1 数据库技术与数据库系统 …… 269
5.1.2 数据描述 …… 274
5.1.3 数据模型 …… 276
5.2 关系代数 …… 280
5.3 数据库设计 …… 286

5.3.1 数据库设计的基本概念 …… 286
5.3.2 数据库设计的过程 …… 287
5.3.3 数据字典 …… 293
习题 …… 295

6 编译技术概述

6.1 编译程序的工作过程及其基本组成…… 297
6.1.1 编译程序的工作过程 …… 297
6.1.2 编译程序的基本组成 …… 298
6.2 状态矩阵法的编译过程…… 299
6.2.1 状态矩阵法的基本原理 …… 299
6.2.2 状态矩阵的压缩 …… 300
6.3 词法分析…… 302
6.3.1 词法分析的任务 …… 302
6.3.2 读字符程序 …… 303
6.3.3 状态矩阵法的词法分析过程 …… 304
6.3.4 算术常数的识别和翻译 …… 307
6.4 中间语言表示…… 310
6.4.1 波兰表示 …… 310
6.4.2 三元组表示 …… 312
6.5 语法的分析与加工…… 316
习题 …… 321

7 应用软件设计与开发技术

7.1 软件工程概述…… 322
7.1.1 软件工程的概念 …… 322
7.1.2 软件生命周期 …… 322
7.1.3 软件支援环境 …… 325
7.2 软件详细设计的表达…… 326
7.2.1 程序流程图 …… 326
7.2.2 NS 图 …… 327
7.2.3 问题分析图 PAD …… 329
7.2.4 判定表 …… 329
7.2.5 过程设计语言 PDL …… 330

7.3 结构化分析与设计方法 …… 331
7.3.1 应用软件开发的原则和方法 …… 331
7.3.2 结构化分析方法 …… 332
7.3.3 结构化设计方法 …… 336
7.4 测试与调试基本技术 …… 342
7.4.1 测试 …… 342
7.4.2 调试 …… 349
7.5 软件开发新技术 …… 351
7.5.1 原型方法 …… 351
7.5.2 瀑布模型 …… 352
7.5.3 面向对象技术 …… 353
习题 …… 354

参考文献 …… 355

第1章 预备知识

1.1 集　　合

1.1.1 集合及其基本运算

1. 集合的基本概念

所谓集合，是指若干个或无穷多个具有相同属性的元(元素)的集体。通常，一个集合名称用大写字母表示，而集合中的某个元素用小写字母表示。

如果集合 M 由 $n(n\geqslant 0)$ 个元素 $a_1,a_2,\cdots,a_n$ 组成，则称集合 M 为有限集。例如，大于 1 而小于 100 的所有整数构成的集合 A 为有限集。如果一个集合中有无穷多个元素，则称此集合为无限集。例如，所有整数构成的集合 Z，所有实数构成的集合 R，大于 0 而小于 1 的所有实数构成的集合 B 等均为无限集。不包括任何元素的集合称为空集。例如，大于 1 而小于 2 的整数构成的集合为空集。空集通常用$\varnothing$表示。

如果 M 是一个集合，a 是集合 M 中的一个元素，则记作 $a\in M$，称元素 a 属于集合 M；如果 a 不是集合 M 中的元素，则记作 $a\notin M$，称元素 a 不属于集合 M。

对于一个集合，通常用以下两种方法表示。

(1) 列举法：用列举法表示一个集合是将此集合中的元素全部列出来，或者列出若干项但能根据规律可知其所有的元素。例如，上述举出的几个集合的例子可以用如下列举法表示：

大于 1 而小于 100 的所有整数的集合 A 可以表示为

$$A=\{2,3,4,\cdots,99\}$$

有限集。

所有整数构成的集合 **Z** 可以表示为

$$\mathbf{Z}=\{0,\pm 1,\pm 2,\pm 3,\cdots\}$$

无限集。

空集表示为

$$\varnothing=\{\}$$

(2) 性质叙述法：用性质叙述法表示一个集合是将集合中的元素所具有的属性描述出来。例如：

大于 1 而小于 100 的所有整数的集合 A 可以表示为

$$A=\{a\mid 1<a<100\text{ 的所有整数}\}$$

所有整数构成的集合 **Z** 可以表示为

$$\mathbf{Z}=\{z\mid z\text{ 为一切整数}\}$$

大于 0 而小于 1 的所有实数构成的集合 B 可以表示为

$$B = \{b \mid 0 < b < 1 \text{ 的所有实数}\}$$

所有实数构成的集合 $\mathbf{R}$ 可以表示为

$$\mathbf{R} = \{r \mid r \text{ 为一切实数}\}$$

设 M 与 N 为两个集合。如果集合 M 中的每一个元素也都为集合 N 的元素，则称集合 M 为 N 的子集，记作 $M \subseteq N$ 或 $N \supseteq M$。如果 $M \subseteq N$，且 N 中至少有一个元素 $a \notin M$，则称 M 是 N 的真子集，记作 $M \subset N$ 或 $N \supset M$。如果 $M \subseteq N$ 且 $N \subseteq M$，则称集合 M 和集合 N 相等，记作 $M = N$。

2. 集合的基本运算

1）两个集合的并(union)

设有两个集合 M 和 N，它们的并集记作 $M \cup N$，其定义如下：

$$M \cup N = \{\alpha \mid \alpha \in M \text{ 或 } \alpha \in N\}$$

即两个集合 M 与 N 的并集是指 M 与 N 中所有元素（去掉重复的元素）组成的集合。

2）两个集合的交(intersection)

设有两个集合 M 和 N，它们的交集记作 $M \cap N$，其定义如下：

$$M \cap N = \{\alpha \mid \alpha \in M \text{ 且 } \alpha \in N\}$$

即两个集合 M 与 N 的交集是指 M 与 N 中所有共同元素组成的集合。

两个集合 M 与 N 的并、交均满足交换律，即

$$M \cup N = N \cup M$$
$$M \cap N = N \cap M$$

3）两个集合的差(difference)

设有两个集合 M 和 N，M 和 N 的差集记作 $M - N$，其定义如下：

$$M - N = \{\alpha \mid \alpha \in M \text{ 但 } \alpha \notin N\}$$

两个集合的差不满足交换律，即

$$M - N \neq N - M$$

例 1.1 设集合

$$A = \{a, b, c, d, e\}$$
$$B = \{d, e, f, g, h\}$$

则

$$A \cup B = \{a, b, c, d, e, f, g, h\}$$
$$A \cap B = \{d, e\}$$
$$A - B = \{a, b, c\}$$
$$B - A = \{f, g, h\}$$

对于集合的并、交、差有以下几个基本性质：

- 结合律

$$(A \cap B) \cap C = A \cap (B \cap C)$$
$$(A \cup B) \cup C = A \cup (B \cup C)$$

- 分配律

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$
$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

- $(A-B)\cup(B-A)=(A\cup B)-(A\cap B)$
- $B\cap(A-B)=\varnothing$

  $(A\cap B)\cup(A-B)=A$

3. 映射

定义 1.1 设 A、B 是两个非空集。如果根据一定的法则 f,对于每一个 $x\in A$,在 B 中都有唯一确定的元素 y 与之对应,则称 f 是定义在 A 上而在 B 中取值的映射,记作 $f: A\to B$,并将 x 与 y 的关系记作 $y=f(x)$。x 称为自变元,y 称为在 f 作用下 x 的像。

集合 A 称为 f 的定义域,$f(A)=\{f(x)\mid x\in A\}$称为 f 的值域。

定义 1.2 设给定映射 $f:A\to B$,且 $B=f(A)$(即 f 的像充满整个 B)。如果对于每个 $y\in B$,仅有唯一的 $x\in A$ 使 $f(x)=y$,则称 f 有逆映射 f^{-1}(它是定义在 $f(A)$上而取值于 A 的映射)。当映射 $f:A\to f(A)$有逆映射时,则称 f 是一一映射。

定义 1.3 若 A、B 两集合有一一映射 f 存在,使 $f(A)=B$,则称 A 与 B 成一一对应。

如果集合 A 与 B 为一一对应,则称它们互相对等,并记作 $A\sim B$。当两个集合互相对等时,称它们有相等的浓度(或相等的元素个数)。

例 1.2 设两集合为

$$A=\{1,2,\cdots,10\}=\{x\mid 1\leqslant x\leqslant 10\text{ 的整数}\}$$
$$B=\{1,2,\cdots,20\}=\{y\mid 1\leqslant y\leqslant 20\text{ 的整数}\}$$

若映射 $f:A\to B$ 为

$$y=2x$$

其中定义域为 A,值域为

$$f(A)=\{2,4,6,\cdots,20\}$$

显然,映射 f 不是一一映射。

例 1.3 设两集合为

$$A=\{x\mid 0\leqslant x\leqslant 4\text{ 的所有实数}\}$$
$$B=\{y\mid 0\leqslant y\leqslant 2\text{ 的所有实数}\}$$

考虑映射 $f:A\to B$

$$y=\sqrt{x}$$

其中定义域为 A,值域为 $f(A)=B$。

显然,f 为一一映射,集合 A 与 B 互相对等,即 $A\sim B$。

集合的对等满足以下性质:

- 自反性。即 $A\sim A$。
- 对称性。即若 $A\sim B$,则 $B\sim A$。
- 传递性。即若 $A\sim B$ 且 $B\sim C$,则 $A\sim C$。

1.1.2 自然数集与数学归纳法

由所有自然数组成的集合

$$\{1,2,3,\cdots\}$$

称为自然数集。自然数集是一个无限集。

由自然数组成的集合均是自然数集的子集。自然数集的子集可以是有限集,也可以是

无限集。

与自然数集对等(即具有相等浓度)的集合称为可列集(或可数集)。任一可列集中的元素排列时可标以正整数下标,即任意可列集 M 均可写成

$$M = \{a_1, a_2, \cdots, a_n, \cdots\}$$

关于自然数集及其子集有以下两个命题成立。

定理 1.1 在自然数集的任一非空子集 M 中,必定有一个最小数。即在集合 M 中有不大于其他任意数的数。

证明:因为 M 非空,所以在 M 中可以取得一自然数 n。

显然,M 中所有不大于 n 的自然数形成的非空子集 N 包含在 M 中,即 $N \subset M$。如果 N 中有最小数,则此最小数就是 M 的最小数。

而在 N 中最多有 n 个自然数($1 \sim n$),因此,N 中有一个最小数。

综上所述,在自然数集的任一非空子集 M 中,必定有一个最小数。定理得证。

定理 1.2 设 M 是由自然数形成的集合,如果它含有 $1,2,\cdots,k$,并且当它含有数 $n-1$, $n-2,\cdots,n-k(n>k)$ 时,也含有数 n,那么它含有所有的自然数,即 M 是自然数集。

证明:设 N 是所有不属于 M 的自然数形成的集合,则 $1,2,\cdots,k \notin N$。

现假设 N 不是空集,则由定理 1.1 可知:在 N 中必定有一个最小数。设此最小数为 c。

由于 c 是 N 中的最小数,即 $c \in N$,因此,$c \notin M$,且 $c \neq 1,2,\cdots,k$,同时,$c-1,c-2,\cdots$, $c-k$ 均为自然数。又由于 c 是 N 中的最小数,所以自然数 $c-1,c-2,\cdots,c-k \notin N$,即 $c-1$, $c-2,\cdots,c-k \in M$,而根据定理中的条件应有 $c \in M$。

由上所述,一方面有 $c \notin M$,另一方面又有 $c \in M$,这就导致矛盾。这个矛盾是由于一开始假定 N 不是空集所造成的。因此,N 只能为空集,即所有自然数均在 M 中,M 为自然数集。

定理得证。

定理 1.2 是数学归纳法的基础。通常,为了证明一个命题对于所有的自然数是真,采用数学归纳法证明的步骤如下:

(1) 证明命题对于自然数 $1,2,\cdots,k$ 是真的。

(2) 假设命题对于自然数 $n-k, n-k+1, \cdots, n-1(n>k)$ 是真的(这一步称为归纳假设)。

(3) 证明命题对于自然数 n 也是真的。

上述步骤(1)中 k 值的选取决定于在步骤(3)的证明过程中要用到归纳假设的最小自然数。在步骤(3)中,为了证明命题对于自然数 n 是真,要用到归纳假设的最小自然数如果为 $n-k$,则在步骤(1)中要对 1 到 k 中的所有自然数证明命题为真。

例 1.4 证明下列命题:

对于任意的自然数 n,必存在一对非负整数(i,j)有

$$7 + n = 3i + 5j$$

下面用数学归纳法证明这个命题。

证明:

(1) 当 $n=1$ 时,有 $7+1=3+5$,即 $i=1, j=1$。命题成立。

当 $n=2$ 时,有 $7+2=3\times3+5\times0$,即 $i=3, j=0$。命题成立。

当 $n=3$ 时，有 $7+3=3\times0+5\times2$，即 $i=0,j=2$。命题成立。

(2) 假设命题对于自然数 $n-1,n-2,n-3(n>3)$ 成立(归纳假设)。

(3) 考虑自然数 n，有

$$7+n=[7+(n-3)]+3$$

根据归纳假设，对于自然数 $n-3$ 命题成立，设存在一对非负整数 i_1,j_1 有

$$7+(n-3)=3i_1+5j_1$$

则有

$$7+n=[7+(n-3)]+3=3(i_1+1)+5j_1=3i+5j$$

其中 $i=i_1+1,j=j_1$ 均为非负整数。即对于自然数 n 命题也成立。

由此得出结论，对于所有的自然数 n 命题成立。

在这个例子中，由于步骤(3)的证明过程中，要用到归纳假设的最小自然数为 $n-3$，因此在步骤(1)中取 $k=3$。

1.1.3 笛卡儿积

在 1.1.1 节中介绍了集合的并、交、差运算。对于集合，还有一种很重要的运算——笛卡儿积(Cartesian Product)。

设有 n 个集合 $D_1,D_2,\cdots,D_n$，此 n 个集合的笛卡儿积定义为

$$D_1\times D_2\times\cdots\times D_n=\{(d_1,d_2,\cdots,d_n)\mid d_i\in D_i,i=1,2,\cdots,n\}$$

其中 $(d_1,d_2,\cdots,d_n)$ 称为 n 元组(n-tuple)，d_i 称为 n 元组的第 i 个分量。

由笛卡儿积的定义可以看出，n 个集合的笛卡儿积是以 n 元组为元素的集合，而每一个 n 元组中的第 i 个分量取自于第 i 个集合 D_i。

例 1.5 设有三个集合

$$A=\{a_1,a_2,a_3\},B=\{b_1,b_2\},C=\{c_1,c_2\}$$

则它们的笛卡儿积为

$$\begin{aligned}A\times B\times C=\{&(a_1,b_1,c_1),(a_1,b_1,c_2),(a_1,b_2,c_1),(a_1,b_2,c_2),\\&(a_2,b_1,c_1)(a_2,b_1,c_2),(a_2,b_2,c_1),(a_2,b_2,c_2),\\&(a_3,b_1,c_1),(a_3,b_1,c_2),(a_3,b_2,c_1),(a_3,b_2,c_2)\}\end{aligned}$$

如果 n 个集合 $D_1,D_2,\cdots,D_n$ 中的元素个数分别为 $m_1,m_2,\cdots,m_n$，则其笛卡儿积中共有 $m_1\times m_2\times\cdots\times m_n$ 个 n 元组。即 n 个集合的笛卡儿积是所有 n 元组组成的集合。

1.1.4 二元关系

定义 1.4 设 M 和 N 是两个集合，则其笛卡儿积

$$M\times N=\{(x,y)\mid x\in M \text{ 且 } y\in N\}$$

的每一个子集称为在 $M\times N$ 上的一个二元关系。

如果 $M=N$，则其笛卡儿积

$$M\times M=\{(x,y)\mid x,y\in M\}$$

的每一个子集称为在集合 M 上的一个二元关系，简称为在集合 M 上的一个关系。

例 1.6 设集合 M 为

$$M=\{a,b,c,d,e,f\}$$

则下列每一个二元组的集合是在集合 M 上的一个关系：

$$R_1 = \{(a,b),(b,c),(c,d),(d,e),(e,f)\}$$

$$R_2 = \{(a,e),(a,a),(c,f),(d,b),(e,a),(f,c),(b,d)\}$$

$$R_3 = \{(a,a),(b,b),(c,c),(d,d),(e,e),(f,f),(c,f),(e,a)\}$$

$$R_4 = \{(a,b),(b,e),(c,d),(d,f),(a,e),(c,f)\}$$

集合 M 上的一个关系实际上反映了集合 M 中各元素之间的联系。

定义 1.5 设 R 是集合 M 上的一个关系。

(1) 如果 $(a,b)\in R$,则称 a 是 b 的关于 R 的前件(predecessor),b 是 a 的关于 R 的后件(successor)。

(2) 如果对于每一个 $a\in M$,都有 $(a,a)\in R$,则称关系 R 是自反的(reflexive);如果对于任何 $a\in M$,$(a,a)\in R$ 均不成立,则称关系 R 是非自反的(antireflexive)。

(3) 如果 $(a,b)\in R$ 时必有 $(b,a)\in R$,则称关系 R 是对称的(symmetric)。

(4) 如果当 $(a,b)\in R$ 且 $(b,c)\in R$ 时必有 $(a,c)\in R$,则称关系 R 是传递的(transitive)。

在例 1.6 中,关系 R_1 是非自反的,但不是对称的,也不是传递的;关系 R_2 是对称的,但不是自反的,不是非自反的,也不是传递的;关系 R_3 是自反的,但不是对称的,也不是传递的;关系 R_4 是传递的,且是非自反的,但不是对称的。

由此可以看出,集合 M 中的各元素之间的逻辑关系可以由集合 M 上的一个关系来描述。

定义 1.6 设 R 是 M 上的一个传递关系,且 $T\subseteq R$。若对于任何 $(x,y)\in R$,在 M 中有元素 $x_0,x_1,x_2,\cdots,x_n(n\geqslant 1)$ 满足:(1) $x_0=x$,(2) $x_n=y$,(3) $(x_{i-1},x_i)\in T(i=1,2,\cdots,n)$,则称关系 T 是关系 R 的基(basis),又称关系 R 是关系 T 的传递体(transitive hull)。

例 1.7 设集合 M 为

$$M = \{1,2,3,4,5\}$$

集合 M 上的一个关系为

$$R = \{(1,2),(2,3),(3,5),(3,4),(1,3),(1,4),(2,4),(2,5),(1,5)\}$$

可以验证,关系 R 是非自反的,且是传递的。现考虑集合 M 上的另一个关系

$$T = \{(1,2),(2,3),(3,5),(3,4)\}$$

显然 $T\subseteq R$,并且可以验证,对于关系 R 中的每一个二元组 (x,y),在 M 中存在元素 $x_0,x_1,x_2,\cdots,x_n$,满足定义 1.6 中的三个条件。因此,T 是 R 的具有 4 个元素(二元组)的基。

同样还可以验证关系

$$T_1 = \{(1,2),(1,4),(2,3),(3,5),(3,4)\}$$

是 R 的具有 5 个元素(二元组)的基。但关系

$$T_2 = \{(1,2),(2,3),(3,4),(4,5)\}$$

不是 R 的基,因为 $(4,5)\notin R$。

1.2 算　　法

1.2.1 算法的基本概念

什么是算法呢? 概括地说,所谓算法是指解题方案的准确而完整的描述。

对于一个问题，如果可以通过一个计算机程序，在有限的存储空间内运行有限长的时间而得到正确的结果，则称这个问题是算法可解的。但算法不等于程序，也不等于计算方法。当然，程序也可以作为算法的一种描述，但程序通常还需考虑很多与方法和分析无关的细节问题，这是因为在编写程序时要受到计算机系统运行环境的限制。通常，程序的编制不可能优于算法的设计。

作为一个算法，一般应具有以下几个基本特征：

1. 能行性(effectiveness)

算法的能行性包括以下两个方面：

(1) 算法中的每一个步骤必须能够实现。如在算法中不允许执行分母为 0 的操作，在实数范围内不能求一个负数的平方根等。

(2) 算法执行的结果要能够达到预期的目的。

针对实际问题设计的算法，人们总是希望能够得到满意的结果。但一个算法又总是在某个特定的计算工具上执行的，因此，算法在执行过程中往往要受到计算工具的限制，使执行结果产生偏差。例如，在进行数值计算时，如果某计算工具具有 7 位有效数字(如程序设计语言中的单精度运算)，则在计算下列三个量

$$A = 10^{12}, B = 1, C = -10^{12}$$

的和时，如果采用不同的运算顺序，就会得到不同的结果，即

$$A + B + C = 10^{12} + 1 + (-10^{12}) = 0$$

$$A + C + B = 10^{12} + (-10^{12}) + 1 = 1$$

而在数学上，$A+B+C$ 与 $A+C+B$ 是完全等价的。因此，算法与计算公式是有差别的。在设计一个算法时，必须要考虑它的能行性，否则是不会得到满意结果的。

2. 确定性(definiteness)

算法的确定性，是指算法中的每一个步骤都必须有明确的定义，不允许有模棱两可的解释，也不允许有多义性。这一性质也反映了算法与数学公式的明显差别。在解决实际问题时，可能会出现这样的情况：针对某种特殊问题，数学公式是正确的，但按此数学公式设计的计算过程可能会使计算机系统无所适从。这是因为根据数学公式设计的计算过程只考虑了正常使用的情况，而当出现异常情况时，此计算过程就不能适应了。

3. 有穷性(finiteness)

算法的有穷性，是指算法必须能在有限的时间内做完，即算法必须能在执行有限个步骤之后终止。数学中的无穷级数，在实际计算时只能取有限项，即计算无穷级数值的过程只能是有穷的。因此，一个数的无穷级数表示只是一个计算公式，而根据精度要求确定的计算过程才是有穷的算法。

算法的有穷性还应包括合理的执行时间的含义。因为，如果一个算法需要执行千万年，显然就失去了实用价值。

4. 拥有足够的情报

一个算法是否有效，还取决于为算法所提供的情报是否足够。通常，算法中的各种运算总是要施加到各个运算对象上，而这些运算对象又可能具有某种初始状态，这是算法执行的起点或是依据。因此，一个算法执行的结果总是与输入的初始数据有关，不同的输入将会有不同的结果输出。当输入不够或输入错误时，算法本身也就无法执行或导致执行有错。一

般来说，当算法拥有足够的情报时，此算法才是有效的；而当提供的情报不够时，算法并不有效。

综上所述，所谓算法，是一组严谨地定义运算顺序的规则，并且每一个规则都是有效的，且是明确的，此顺序将在有限的次数内终止。

1.2.2 算法设计基本方法

本节介绍工程上常用的几种算法设计方法。在实际应用时，各种方法之间往往存在着一定的联系。

1. 列举法

列举法的基本思想是，根据提出的问题，列举所有可能的情况，并用问题中给定的条件检验哪些是需要的，哪些是不需要的。因此，列举法常用于解决"是否存在"或"有多少种可能"等类型的问题，例如求解不定方程的问题。

列举法的特点是算法比较简单。但当列举的可能情况较多时，执行列举算法的工作量将会很大。因此，在用列举法设计算法时，使方案优化，尽量减少运算工作量，是应该重点注意的。通常，在设计列举算法时，只要对实际问题进行详细的分析，将与问题有关的知识条理化、完备化、系统化，从中找出规律；或对所有可能的情况进行分类，引出一些有用的信息，是可以大大减少列举量的。

下面举例说明利用列举算法解决问题时如何对算法进行优化。

例 1.8 设每只母鸡值 3 元，每只公鸡值 2 元，两只小鸡值 1 元。现要用 100 元钱买 100 只鸡，设计买鸡方案。

假设买母鸡 I 只，公鸡 J 只，小鸡 K 只。根据题意，粗略的列举算法用 C++ 描述如下：

```
//baiji1.cpp
    #include<iostream>
    using namespace std;
    int main()
    {
        int i, j, k;
        for (i=0; i<=100; i++)
        for (j=0; j<=100; j++)
        for (k=0; k<=100; k++)
        {
            if ((i+j+k==100) && (3*i+2*j+0.5*k==100.0))
                cout<<i<<"  "<<j<<"  "<<k<<endl;
        }
      return 0;
    }
```

在这个算法中，共嵌套有三层循环，每层循环各需要循环 101 次，因此，总循环次数为 101^3。但只要对问题进行分析，就会发现对这个算法还可进行优化，减少大量不必要的循环次数。

首先，考虑到母鸡为 3 元一只，因此，母鸡最多只能买 33 只，即算法中的外循环没有必

要从 0 到 100，而只需要从 0 到 33 就可以了。

其次，考虑到公鸡为 2 元一只，因此，公鸡最多只能买 50 只。又考虑到对公鸡的列举是在算法的第二层循环中，此时已经买了 I 只母鸡，且买一只母鸡的价钱相当于买 1.5 只公鸡。因此，由第一层循环已经确定买 I 只母鸡的前提下，公鸡最多只能买 $50-1.5I$ 只，即第二层对 J 的循环只需从 0 到 $50-1.5I$ 只就可以了。

最后，考虑到买的总鸡数为 100，而由第一层循环已确定买 I 只母鸡，由第二层循环已确定买 J 只公鸡，因此，买小鸡的数量只能是 $K=100-I-J$ 只，即第三层循环已经没有必要了。

经过以上分析，可以将上述算法进行修改。经修改后的算法用 C++ 描述如下：

```
//baiji2.cpp
    #include<iostream>
    using namespace std;
    int main()
    {
        int i, j, k;
        for (i=0; i<=33; i++)
        for (j=0; j<=50-1.5*i; j++)
        {
            k=100-i-j;
            if (3*i+2*j+0.5*k==100.0)
                cout<<i<<"  "<<j<<"  "<<k<<endl;
        }
        return 0;
    }
```

不难分析，经修改后的算法的列举量（即循环次数）为

$$\sum_{I=0}^{33}(51-1.5I)\approx 894$$

这个程序的运行结果如下：

```
 2  30  68
 5  25  70
 8  20  72
11  15  74
14  10  76
17   5  78
20   0  80
```

列举原理是计算机应用领域中十分重要的原理。许多实际问题，若采用人工列举是不可想象的，但由于计算机的运算速度快，擅长重复操作，可以很方便地进行大量列举。列举算法虽然是一种比较笨拙而原始的方法，其运算量比较大，但在有些实际问题中（如寻找路径、查找、搜索等问题），局部使用列举法却是很有效的。因此，列举算法是计算机算法中的一个基础算法。

2. 归纳法

归纳法的基本思想是,通过列举少量的特殊情况,经过分析,最后找出一般的关系。显然,归纳法要比列举法更能反映问题的本质,并且可以解决列举量为无限的问题。但是,从一个实际问题中总结归纳出一般的关系,并不是一件容易的事情,尤其是要归纳出一个数学模型更为困难。从本质上讲,归纳就是通过观察一些简单而特殊的情况,最后总结出有用的结论或解决问题的有效途径。

归纳是一种抽象,即从特殊现象中找出一般关系。但由于在归纳的过程中不可能对所有的情况进行列举,因此,最后由归纳得到的结论还只是一种猜测,还需要对这种猜测加以必要的证明。实际上,通过精心观察得到的猜测最后得不到证实或证明是错的,也是常有的事。

3. 递推

所谓递推,是指从已知的初始条件出发,逐次推出所要求的各中间结果和最后结果。其中初始条件或是问题本身已经给定,或是通过对问题的分析与化简而得到确定。递推本质上也属于归纳法,工程上许多递推关系式实际上是通过对实际问题的分析与归纳而得到的,因此,递推关系式往往是归纳的结果。

递推算法在数值计算中是极为常见的。但是,对于数值型的递推算法必须要注意数值计算的稳定性问题。

4. 递归

人们在解决一些复杂问题时,为了降低问题的复杂程度(如问题的规模等),一般总是将问题逐层分解,最后归结为一些最简单的问题。这种将问题逐层分解的过程,实际上并没有对问题进行求解,而是在解决了最后那些最简单的问题后,再沿着原来分解的逆过程逐步进行综合,这就是递归的基本思想。由此可以看出,递归的基础也是归纳。在工程实际中,有许多问题就是用递归来定义的,数学中的许多函数也是用递归来定义的。递归在可计算性理论和算法设计中占有很重要的地位。

下面用一个简单的例子来说明递归的基本思想。

例 1.9 编写一个过程,对于输入的参数 n,依次打印输出自然数 $1\sim n$。

这是一个很简单的问题,实际上不用递归就能解决。对应的算法用 C++ 描述如下:

```
#include<iostream>
using namespace std;
void wrt(int  n)
{ int  k;
  for (k=1; k<=n; k++)  cout<<k<<endl;
  return;
}
```

这个问题还可以用递归来解决。对应的算法用 C++ 描述如下:

```
#include<iostream>
using namespace std;
void wrt1(int n)
{ if (n!=0)
```

```
    { wrt1(n-1);  cout<<n<<endl; }
  return;
}
```

在递归函数 wrt1()中，n 是形参。在开始执行函数 wrt1()时，首先要判断形参变量值(开始时为 n)是否不等于 0，如果不等于 0，则将形参值减 1(即 $n-1$)后作为新的实参再调用函数 wrt1()；在调用函数 wrt1()时，又需判断形参值(此时已变为 $n-1$)是否不等于 0，如果不等于 0，则又将形参值减 1(即 $n-2$)后作为新的实参再次调用函数 wrt1()……这个过程一直进行下去，直到函数 wrt1()的形参值等于 0 为止。此时，由于在先前各层的函数调用中，函数 wrt1()实际上没有执行完，即各层中的形参值还没有被打印输出，这就需要逐层返回，以便打印输出各层中的输入参数 $1,2,\cdots,n$。为此，在递归算法的执行过程中，需要记忆各层调用中的参数，以便在逐层返回时恢复这些参数继续进行处理。具体来说，在函数 wrt1()开始执行后，随着各次的递归调用，逐次记忆各层调用中的输入参数 $n,n-1,n-2,\cdots,2,1$，在逐层返回时，又依次(按记忆的相反次序)将这些参数打印输出。

在程序设计中，递归是一个很有用的工具。对于一些比较复杂的问题，设计成递归算法可使结构清晰，可读性也强。

由上例可以看出，自己调用自己的过程称为递归调用过程。

递归分为直接递归与间接递归两种。如果一个算法 P 显式地调用自己则称为直接递归。例如上述递归函数 wrt1()是一个直接递归的算法。如果算法 P 调用另一个算法 Q，而算法 Q 又调用算法 P，则称为间接递归调用。

递归是一种很重要的算法设计方法之一。实际上，递归过程能将一个复杂的问题归结为若干个较简单的问题。然后将这些较简单的每一个问题再归结为更简单的问题，这个过程可以一直做下去，直到最简单的问题为止。

有些实际问题，既可以归纳为递推算法，又可以归纳为递归算法。递推与递归的实现方法是大不一样的。递推是从初始条件出发，逐次推出所需求的结果；而递归则是从算法本身到达递归边界的。通常，递归算法要比递推算法清晰易读，其结构比较简练。特别是在许多比较复杂的问题中，很难找到从初始条件推出所需结果的全过程，此时，设计递归算法要比递推算法容易得多，但递归算法的执行效率比较低。

5. 减半递推技术

解决实际问题的复杂程度往往与问题的规模有着密切的关系，因此，降低问题的规模是算法设计的关键，而分治法是降低问题规模的一种有效的方法。所谓分治法，即对问题分而治之。工程上常用的分治法是减半递推技术。这个技术在快速算法的研究中有很重要的实用价值。

所谓“减半”，是指将问题的规模减半，而问题的性质不变。所谓“递推”，是指重复“减半”的过程。

下面举例说明利用减半递推技术设计算法的基本思想。

例 1.10 设方程 $f(x)=0$ 在区间$[a,b]$上有实根，且 $f(a)$与 $f(b)$异号。利用二分法求该方程在区间$[a,b]$上的一个实根。

二分法求方程实根的减半递推过程如下：

首先取给定区间的中点 $c=(a+b)/2$。

然后判断 $f(c)$ 是否为 0。若 $f(c)=0$，则说明 c 即为所求的根，求解过程结束；如果 $f(c)\neq 0$，则根据以下原则将原区间减半。

若 $f(a)f(c)<0$，则取原区间的前半部分；若 $f(b)f(c)<0$，则取原区间的后半部分。

最后判断减半后的区间长度是否已经很小。

若 $|a-b|<\varepsilon$，则过程结束，取 $(a+b)/2$ 为根的近似值；若 $|a-b|\geqslant\varepsilon$，则重复上述的减半过程。

这个算法留给读者自己去描述。

6. 回溯法

前面讨论的递推和递归算法本质上是对实际问题进行归纳的结果，而减半递推技术也是归纳法的一个分支。在工程上，有些实际问题却很难归纳出一组简单的递推公式或直观的求解步骤，并且也不能进行无限列举。对于这类问题，一种有效的方法是“试”。通过对问题的分析，找出一个解决问题的线索，然后沿着这个线索逐步试探。对于每一步试探，若试探成功，就得到问题的解；若试探失败，就逐步回退，换别的路线再进行试探。这种方法称为回溯法。回溯法在处理复杂数据结构方面有着广泛的应用。

下面举例说明回溯法的基本思想。

例 1.11 求解皇后问题。

由 n^2 个方块排成 n 行 n 列的正方形称为“n 元棋盘”。如果两个皇后位于棋盘上的同一行或同一列或同一对角线上，则称它们为互相攻击。现要求找使 n 元棋盘上的 n 个皇后互不攻击的所有布局。

n 个皇后在 n 元棋盘上的布局共有 n^n 种，为了从中找出互不攻击的布局，需要对此 n^n 种方案逐个进行检查，将有攻击的布局剔除掉。这是一种列举法。但这种方法对于较大的 n，其运算量会急剧增加。而在实际上，逐一列举也是没有必要的。例如，如果第一行上的皇后在第一列，则第二行上的皇后就不可能在第一列。

下面用回溯法来求解皇后问题。

假设棋盘上的每一行放置一个皇后，分别用自然数进行编号为 $1,2,\cdots,n$。

首先定义一个长度为 n 的一维数组 q，其中每一个元素 $q[i]$$(i=1,2,\cdots,n)$ 随时记录第 i 行上的皇后所在的列号。

初始时，先将各皇后放在各行的第 1 列。即数组 q 的初值为

$$q[i]=1, i=1,2,\cdots,n$$

第 i 行与第 j 行上的皇后在某一对角线上的条件为

$$|q[i]-q[j]|=|i-j|$$

而它们在同一列上的条件为

$$q[i]=q[j]$$

回溯法的步骤如下：

从第一行(即 $i=1$)开始进行以下过程。

设前 $i-1$ 行上的皇后已经布局好，即它们均互不攻击。现在考虑安排第 i 行上的皇后的位置，使之与前 $i-1$ 行上的皇后也都互不攻击。为了实现这一目的，可以从第 i 行皇后的当前位置 $q[i]$ 开始按如下规则向右进行搜索：

(1) 若 $q[i]=n+1$，则说明第 i 个皇后暂时无法安排，将第 i 个皇后放在第 1 列(即置

$q[i]=1$)，且回退一行，考虑重新安排第 $i-1$ 行上的皇后与前 $i-2$ 行上的皇后均互不攻击的下一个位置。此时如果已退到第 0 行(实际没有这一行)，则过程结束。

(2) 若 $q[i]\leqslant n$，则需检查第 i 行上的皇后与前 $i-1$ 行的皇后是否互不攻击。若有攻击，则将第 i 行上的皇后右移一个位置(即置 $q[i]=q[i]+1$)，重新进行这个过程；若无攻击，则考虑安排下一行上的皇后位置，即置 $i=i+1$。

(3) 若当前安排好的皇后是在最后一行(即第 n 行)，则说明已经找到了 n 个皇后互不攻击的一个布局，将这个布局输出(即输出 $q[i]$，$i=1,2,\cdots,n$)。然后将第 n 行上的皇后右移一个位置(即置 $q[n]=q[n]+1$)，重新进行这个过程，以便寻找下一个布局。

以上介绍了几种工程上常用的算法设计的基本方法。实际上，算法设计的方法还有很多，如数字模拟法、用于数值近似的数值法等，在此不再一一介绍了，有兴趣的读者可参看有关算法方面的著作。

1.2.3 算法的复杂度分析

算法的复杂度主要包括时间复杂度和空间复杂度。

1. 算法的时间复杂度

所谓算法的时间复杂度，是指执行算法所需要的计算工作量。

为了能够比较客观地反映出一个算法的效率，在度量一个算法的工作量时，不仅应该与所使用的计算机、程序设计语言以及程序编制者无关，而且还应该与算法实现过程中的许多细节无关。为此，可以用算法在执行过程中所需基本运算的执行次数来度量算法的工作量。基本运算反映了算法运算的主要特征，因此，用基本运算的次数来度量算法工作量是客观的也是实际可行的，有利于比较同一问题的几种算法的优劣。例如，在考虑两个矩阵相乘时，可以将两个实数之间的乘法运算作为基本运算，而对于所用的加法(或减法)运算可以忽略不计。又如，当需要在一个表中进行查找时，可以将两个元素之间的比较作为基本运算。

算法所执行的基本运算次数还与问题的规模有关。例如，两个 20 阶矩阵相乘与两个 10 阶矩阵相乘，所需要的基本运算(即两个实数的乘法)次数显然是不同的，前者需要更多的运算次数。因此，在分析算法的工作量时，还必须对问题的规模进行度量。

综上所述，算法的工作量用算法所执行的基本运算次数来度量，而算法所执行的基本运算次数是问题规模的函数，即

$$\text{算法的工作量} = f(n)$$

其中 n 是问题的规模。例如，两个 n 阶矩阵相乘所需要的基本运算(即两个实数的乘法)次数为 n^3，即计算工作量为 n^3，也就是时间复杂度为 n^3。

在具体分析一个算法的工作量时，还会存在这样的问题：对于一个固定的规模，算法所执行的基本运算次数还可能与特定的输入有关，而实际上又不可能将所有可能情况下算法所执行的基本运算次数都列举出来。例如，“在长度为 n 的一维数组中查找值为 x 的元素”，若采用顺序搜索法，即从数组的第一个元素开始，逐个与被查值 x 进行比较。显然，如果第一个元素恰为 x，则只需要比较 1 次。但如果 x 为数组的最后一个元素，或者 x 不在数组中，则需要比较 n 次才能得到结果。因此，在这个问题的算法中，其基本运算(即比较)的次数与具体的被查值 x 有关。

在同一问题规模下，如果算法执行所需的基本运算次数取决于某一特定输入时，可以用以下两种方法来分析算法的工作量。

1）平均性态（Average Behavior）

所谓平均性态分析，是指用各种特定输入下的基本运算次数的带权平均值来度量算法的工作量。

设 x 是所有可能输入中的某个特定输入，$p(x)$ 是 x 出现的概率（即输入为 x 的概率），$t(x)$ 是算法在输入为 x 时所执行的基本运算次数，则算法的平均性态定义为

$$A(n) = \sum_{x \in D_n} p(x)t(x)$$

其中 D^n 表示当规模为 n 时，算法执行时所有可能输入的集合。这个式子中的 $t(x)$ 可以通过分析算法来加以确定；而 $p(x)$ 必须由经验或用算法中有关的一些特定信息来加以确定，通常是不能解析地加以计算的。如果确定 $p(x)$ 比较困难，则会给平均性态的分析带来困难。

2）最坏情况复杂性（Worst-Case Complexity）

所谓最坏情况分析，是指在规模为 n 时，算法所执行的基本运算的最大次数。它定义为

$$W(n) = \max_{x \in D_n}\{t(x)\}$$

显然，$W(n)$ 的计算要比 $A(n)$ 的计算方便得多。由于 $W(n)$ 实际上是给出了算法工作量的一个上界，因此，它比 $A(n)$ 更具有实用价值。

下面通过一个例子来说明算法复杂度的平均性态分析与最坏情况分析。

例 1.12 采用顺序搜索法，在长度为 n 的一维数组中查找值为 x 的元素。即从数组的第一个元素开始，逐个与被查值 x 进行比较。基本运算为 x 与数组元素的比较。

首先考虑平均性态分析。

设被查项 x 在数组中的概率为 q。当需要查找的 x 为数组中第 i 个元素时，则在查找过程中需要做 i 次比较。当需要查找的 x 不在数组中时（即数组中没有 x 这个元素），则需要和数组中所有的元素进行比较。即

$$t_i = \begin{cases} i, & 1 \leqslant i \leqslant n \\ n, & i = n+1 \end{cases}$$

其中 $i=n+1$ 表示 x 不在数组中的情况。

如果假设需要查找的 x 出现在数组中每个位置上的可能性是一样的，则 x 出现在数组中每一个位置上的概率为 q/n（因为前面已经假设 x 在数组中的概率为 q），而 x 不在数组中的概率为 $1-q$。即

$$p_i = \begin{cases} q/n, & 1 \leqslant i \leqslant n \\ 1-q, & i = n+1 \end{cases}$$

其中 $i=n+1$ 表示 x 不在数组中的情况。

因此，用顺序搜索法在长度为 n 的一维数组中查找值为 x 的元素时，在平均情况下需要的比较次数为

$$A(n) = \sum_{i=1}^{n+1} p_i t_i = \sum_{i=1}^{n} (q/n)i + (1-q)n = (n+1)q/2 + (1-q)n$$

如果已知需要查找的 x 一定在数组中，此时 $q=1$，则 $A(n)=(n+1)/2$。这就是说，在这种情况下，用顺序搜索法在长度为 n 的一维数组中查找值为 x 的元素时，在平均情况下需要检查数组中一半的元素。

如果已知需要查找的 x 有一半的机会在数组中，此时 $q=1/2$，则

$$A(n) = [(n+1)/4] + n/2 \approx 3n/4$$

这就是说，在这种情况下，用顺序搜索法在长度为 n 的一维数组中查找值为 x 的元素，在平均情况下需要检查数组中 3/4 的元素。

再考虑最坏情况分析。

在这个例子中，最坏情况发生在需要查找的 x 是数组中的最后一个元素或 x 不在数组中的时候，此时显然有

$$W(n) = \max\{t_i \mid 1 \leqslant i \leqslant n+1\} = n$$

在上述例子中，算法执行的工作量是与具体的输入有关的，$A(n)$只是它的加权平均值。实际上对于某个特定的输入，其计算工作量未必是 $A(n)$，且 $A(n)$也不一定等于 $W(n)$。在另外一些情况下，算法的计算工作量与输入无关，即当规模为 n 时，在所有可能的输入下，算法所执行的基本运算次数是一定的，此时有 $A(n)=W(n)$。例如，两个 n 阶的矩阵相乘，都需要做 n^3 次实数乘法，而与输入矩阵的具体元素无关。

2. 算法的空间复杂度

一个算法的空间复杂度，一般是指执行这个算法所需要的内存空间。

一个算法所占用的存储空间包括算法程序所占的空间、输入的初始数据所占的存储空间以及算法执行过程中所需要的额外空间。其中额外空间包括算法程序执行过程中的工作单元以及某种数据结构所需要的附加存储空间（例如，在链式结构中，除了要存储数据本身外，还需要存储链接信息）。如果额外空间量相对于问题规模来说是常数，则称该算法是原地(in place)工作的。在许多实际问题中，为了减少算法所占的存储空间，通常采用压缩存储技术，以便尽量减少不必要的额外空间。

习　题

1.1　设集合 $M=\{d_1,d_2,d_3,d_4,d_5,d_6\}$上的一个关系 R 如下：

$$R = \{(d_1,d_2),(d_2,d_4),(d_5,d_4),(d_1,d_5),\\ (d_1,d_4),(d_1,d_6),(d_1,d_3),(d_3,d_6)\}$$

试验证

$$T_1 = \{(d_1,d_2),(d_2,d_4),(d_1,d_3),(d_3,d_6),(d_5,d_4),(d_1,d_5)\}$$

是 R 的具有 6 个元素的基，而

$$T_2 = \{(d_2,d_4),(d_1,d_3),(d_1,d_2),(d_6,d_5)\}$$

不是 R 的基。

1.2　设给定三个整数 a,b,c，试写出寻找其中数的一个算法（用 C/C++ 描述），并分析在平均情况与最坏情况下，算法分别要进行多少次比较？

1.3 利用减半递推技术，写出求长度为 n 的数组中最大元素的递归算法（用 C/C++ 描述）。设 $n=2^k$，其中 $k\geqslant 1$。

1.4 编写二分法求方程实根的减半递推算法（用 C/C++ 描述）。

1.5 编写用回溯法求解皇后问题的算法（用 C/C++ 描述）。

1.6 设有 12 个小球。其中 11 个小球的重量相同，称为好球；有一个小球的重量与 11 个好球的重量不同（或轻或重），称这个小球为坏球。试编写一个算法（用 C/C++ 描述），用一个无砝码的天平称 3 次找出这个坏球，并确定其比好球轻还是重。

第 2 章　基本数据结构及其运算

利用计算机进行数据处理是计算机应用的重要领域。在进行数据处理时，实际需要处理的数据元素一般有很多，而这些数据元素都需要存放在计算机中，因此，大量的数据元素在计算机中如何组织，以便提高数据处理的效率，并且节省计算机的存储空间，这是进行数据处理的关键问题。

显然，杂乱无章的数据是不便于处理的。而将大量的数据随意地存放在计算机中，实际上也是“自找苦吃”，对数据处理更是不利。

数据结构作为计算机的一门学科，主要研究和讨论以下三个方面的问题：

（1）数据集合中各数据元素之间所固有的逻辑关系，即数据的逻辑结构。

（2）在对数据进行处理时，各数据元素在计算机中的存储关系，即数据的存储结构。

（3）对各种数据结构进行的运算。

讨论以上各问题的主要目的是为了提高数据处理的效率。所谓提高数据处理的效率，主要包括两个方面：一是提高数据处理的速度，二是尽量节省在数据处理过程中所占用的计算机存储空间。

本章主要讨论工程上实用的一些基本数据结构，它们是软件设计的基础。

2.1　数据结构的基本概念

计算机已被广泛用于数据处理。实际问题中的各数据元素之间总是相互关联的。所谓数据处理，是指对数据集合中的各元素以各种方式进行运算，包括插入、删除、查找、更改等运算，也包括对数据元素进行分析。在数据处理领域中，建立数学模型有时并不十分重要，事实上，许多实际问题是无法表示成数学模型的。人们最感兴趣的是知道数据集合中各数据元素之间存在什么关系，为了提高处理效率，应如何组织它们，即如何表示所需要处理的数据元素。

2.1.1　什么是数据结构

简单地说，数据结构是指相互有关联的数据元素的集合。例如，向量和矩阵就是数据结构，在这两个数据结构中，数据元素之间有着位置上的关系。又如，图书馆中的图书卡片目录，则是一个较为复杂的数据结构，对于列在各卡片上的各种书之间，可能在主题、作者等问题上相互关联，甚至一本书本身也有不同的相关成分。

数据元素具有广泛的含义。一般来说，现实世界中客观存在的一切个体都可以是数据元素。例如：

描述一年四季的季节名

春，夏，秋，冬

可以作为季节的数据元素。

表示数值的各个数

18,11,35,23,16…

可以作为数值的数据元素。

表示家庭成员的各成员名

父亲,儿子,女儿

可以作为家庭成员的数据元素。甚至每一个客观存在的事件,如一次演出、一次借书、一次比赛等也可以作为数据元素。

总之,在数据处理领域中,每一个需要处理的对象都可以抽象成数据元素。数据元素一般简称为元素。

在实际应用中,被处理的数据元素一般有很多,而且,作为某种处理,其中的数据元素一般具有某种共同特征。例如,{春,夏,秋,冬}这四个数据元素有一个共同特征,即它们都是季节名,分别表示了一年中的四个季节,从而这四个数据元素构成了季节名的集合。又如,{父亲,儿子,女儿}这三个数据元素也有一个共同特征,即它们都是家庭的成员名,从而构成了家庭成员名的集合。一般来说,人们不会同时处理特征完全不同且互相之间没有任何关系的各类数据元素,对于具有不同特征的数据元素总是分别进行处理。

一般情况下,在具有相同特征的数据元素集合中,各个数据元素之间存在有某种关系(即联系),这种关系反映了该集合中的数据元素所固有的一种结构。在数据处理领域中,通常把数据元素之间这种固有的关系简单地用前后件关系(或直接前驱与直接后继关系)来描述。

例如:在考虑一年四个季节的顺序关系时,则“春”是“夏”的前件(即直接前驱,下同),而“夏”是“春”的后件(即直接后继,下同)。同样,“夏”是“秋”的前件,“秋”是“夏”的后件;“秋”是“冬”的前件,“冬”是“秋”的后件。

在考虑家庭成员间的辈分关系时,则“父亲”是“儿子”和“女儿”的前件,而“儿子”与“女儿”都是“父亲”的后件。

前后件关系是数据元素之间的一个基本关系,但前后件关系所表示的实际意义是随具体对象的不同而不同。一般来说,数据元素之间的任何关系都可以用前后件关系来描述。

1. 数据的逻辑结构

前面提到,数据结构是指反映数据元素之间关系的数据元素集合的表示。更通俗地说,数据结构是指带有结构的数据元素的结合。在此,所谓结构实际上就是指数据元素之间的前后件关系。

由上所述,一个数据结构应包含以下两方面的信息:

(1) 表示数据元素的信息。

(2) 表示各数据元素之间的前后件关系。

在以上所述的数据结构中,其中数据元素之间的前后件关系是指它们的逻辑关系,而与它们在计算机中的存储位置无关。因此,上面所述的数据结构实际上是数据的逻辑结构。

所谓数据的逻辑结构,是指反映数据元素之间逻辑关系的数据结构。

由前面的叙述可以知道，数据的逻辑结构有两个要素：一是数据元素的集合，通常记为D；二是D上的关系，它反映了D中各数据元素之间的前后件关系，通常记为R。即一个数据结构可以表示成

$$B = (D,R)$$

其中B表示数据结构。为了反映D中各数据元素之间的前后件关系，一般用二元组来表示。例如，假设a与b是D中的两个数据，则二元组(a,b)表示a是b的前件，b是a的后件。这样，在D中的每两个元素之间的关系都可以用这种二元组来表示。

例 2.1 一年四季的数据结构可以表示成

$$B = (D,R)$$
$$D = \{\text{春},\text{夏},\text{秋},\text{冬}\}$$
$$R = \{(\text{春},\text{夏}),(\text{夏},\text{秋}),(\text{秋},\text{冬})\}$$

例 2.2 家庭成员数据结构可以表示成

$$B = (D,R)$$
$$D = \{\text{父亲},\text{儿子},\text{女儿}\}$$
$$R = \{(\text{父亲},\text{儿子}),(\text{父亲},\text{女儿})\}$$

例 2.3 n维向量

$$\boldsymbol{X} = (x_1,x_2,\cdots,x_n)$$

也是一种数据结构。即$X=(D,R)$，其中数据元素的集合为

$$D = \{x_1,x_2,\cdots,x_n\}$$

关系为

$$R = \{(x_1,x_2),(x_2,x_3),\cdots,(x_{n-1},x_n)\}$$

对于一些复杂的数据结构来说，它的数据元素可以是另一种数据结构。

例如，$m\times n$的矩阵

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

是一个数据结构。在这个数据结构中，矩阵的每一行

$$A_i = (a_{i1},a_{i2},\cdots,a_{in}),i = 1,2,\cdots,m$$

可以看成是它的一个数据元素。即这个数据结构的数据元素的集合为

$$D = \{A_1,A_2,\cdots,A_m\}$$

D上的一个关系为

$$R = \{(A_1,A_2),(A_2,A_3),\cdots,(A_{m-1},A_m)\}$$

显然，数据结构A中的每一个数据元素$A_i(i=1,2,\cdots,m)$又是另一个数据结构，即数据元素的集合为

$$D_i = \{a_{i1},a_{i2},\cdots,a_{in}\}$$

D_i上的一个关系为

$$R_i = \{(a_{i1},a_{i2}),(a_{i2},a_{i3}),\cdots,(a_{i,n-1},a_{in})\}$$

2. 数据的存储结构

数据处理是计算机应用的一个重要领域，在实际进行数据处理时，被处理的各数据元素总是被存放在计算机的存储空间中，并且，各数据元素在计算机存储空间中的位置关系与它们的逻辑关系不一定是相同的，而且一般也不可能相同。例如，在前面提到的一年四个季节的数据结构中，“春”是“夏”的前件，“夏”是“春”的后件。在对它们进行处理时，在计算机存储空间中，“春”这个数据元素的信息不一定被存储在“夏”这个数据元素信息的前面，而可能在后面，也可能不是紧邻在前面，而是中间被其他信息所隔开。又如，在家庭成员的数据结构中，“儿子”和“女儿”都是“父亲”的后件。在计算机存储空间中，根本不可能将“儿子”和“女儿”这两个数据元素的信息都紧邻存放在“父亲”这个数据元素信息的后面，即在存储空间中与“父亲”紧邻的只可能是其中的一个。由此可以看出，一个数据结构中的各数据元素在计算机存储空间中的位置关系与逻辑关系是有可能不同的。

数据的逻辑结构在计算机存储空间中的存放形式称为数据的存储结构（也称数据的物理结构）。

由于数据元素在计算机存储空间中的位置关系可能与逻辑关系不同，因此，为了表示存放在计算机存储空间中的各数据元素之间的逻辑关系（即前后件关系），在数据的存储结构中，不仅要存放各数据元素的信息，还需要存放各数据元素之间的前后件关系的信息。

一般来说，一种数据的逻辑结构根据需要可以表示成多种存储结构，常用的存储结构有顺序、链接、索引等存储结构。采用不同的存储结构，其数据处理的效率是不同的。因此，在进行数据处理时，选择合适的存储结构是很重要的。

2.1.2 数据结构的图形表示

一个数据结构除了用二元关系表示外，还可以直观地用图形表示。在数据结构的图形表示中，对于数据集合 D 中的每一个数据元素用中间标有元素值的方框表示，一般称之为数据结点，并简称为结点；为了进一步表示各数据元素之间的前后件关系，对于关系 R 中的每一个二元组，用一条有向线段从前件结点指向后件结点。

例如，一年四季的数据结构可以用如图 2.1 所示的图形来表示。又如，反映家庭成员间辈分关系的数据结构可以用如图 2.2 所示的图形表示。

图 2.1　一年四季数据结构的图形表示

显然，用图形方式表示一个数据结构是很方便的，并且也比较直观。有时在不会引起误会的情况下，在前件结点到后件结点连线上的箭头可以省去。例如，在图 2.2 中，即使将“父亲”结点与“儿子”结点连线上的箭头以及“父亲”结点与“女儿”结点连线上的箭头都去掉，也同样表示了“父亲”是“儿子”与“女儿”的前件，“儿子”与“女儿”均是“父亲”的后件，而不会引起误会。

例 2.4　用图形表示数据结构 $B=(D,R)$，其中

$$D = \{d_i \mid 1 \leqslant i \leqslant 7\} = \{d_1, d_2, d_3, d_4, d_5, d_6, d_7\}$$

$$R = \{(d_1, d_3), (d_1, d_7), (d_2, d_4), (d_3, d_6), (d_4, d_5)\}$$

这个数据结构的图形表示如图 2.3 所示。

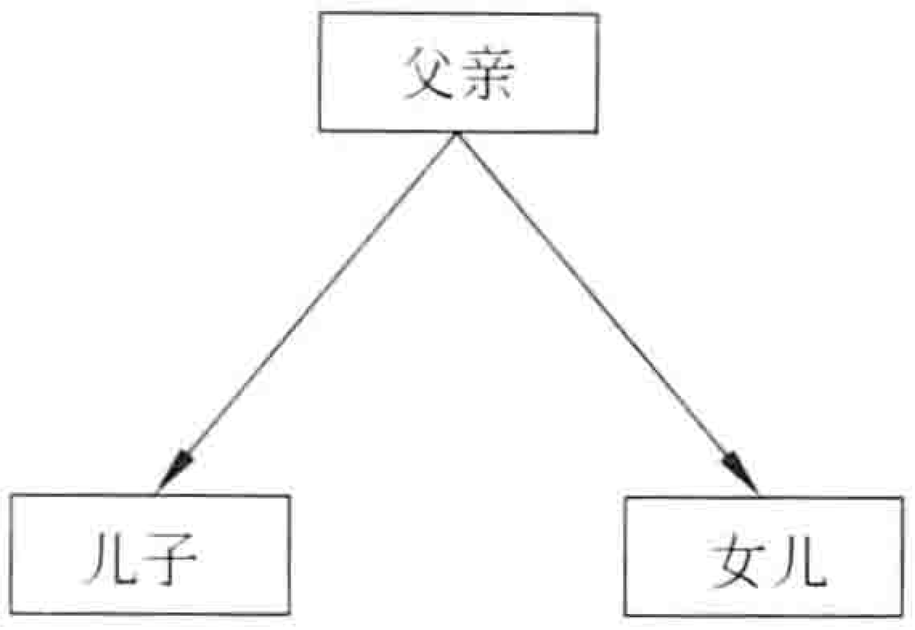


图 2.2 家庭成员间辈分关系数据结构的图形表示

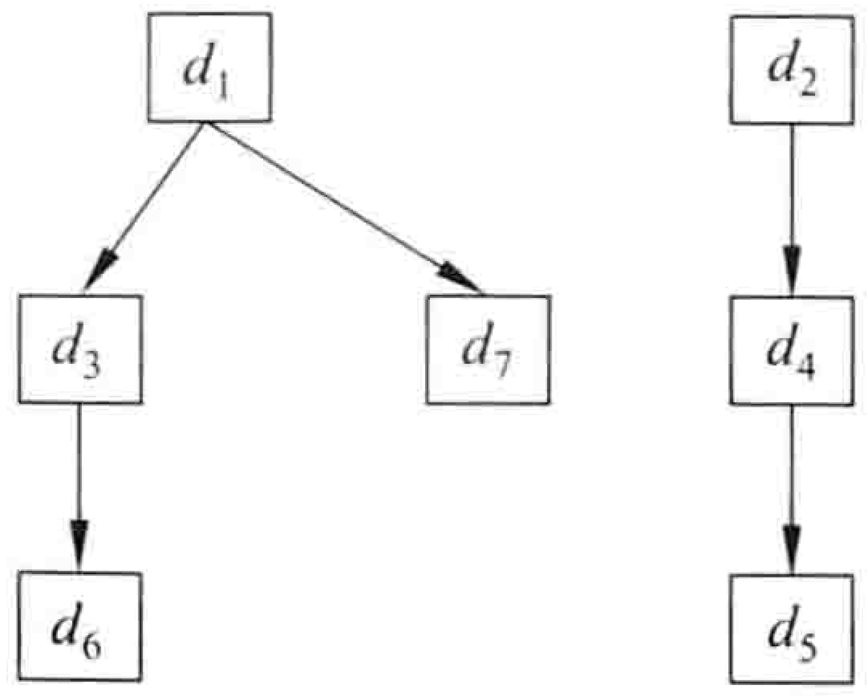


图 2.3 例 2.4 数据结构的图形表示

在数据结构中，没有前件的结点称为根结点；没有后件的结点称为终端结点（也称为叶子结点）。例如，在图 2.1 所示的数据结构中，元素“春”所在的结点（简称为结点“春”，下同）为根结点，结点“冬”为终端结点；在图 2.2 所示的数据结构中，结点“父亲”为根结点，结点“儿子”与“女儿”均为终端结点；在图 2.3 所示的数据结构中，有两个根结点 d_1 与 d_2，有三个终端结点 d_6、d_7、d_5。数据结构中除了根结点与终端结点外的其他结点一般称为内部结点。

通常，一个数据结构中的元素结点可能是在动态变化的。根据需要或在处理过程中，可以在一个数据结构中增加一个新结点（称为插入运算），也可以删除数据结构中的某个结点（称为删除运算）。插入与删除是对数据结构的两种基本运算。除此之外，对数据结构的运算还有查找、分类、合并、分解、复制和修改等。在对数据结构的处理过程中，不仅数据结构中的结点（即数据元素）个数在动态地变化，而且，各数据元素之间的关系也有可能在动态地变化。例如，一个无序表可以通过排序处理而变成有序表；一个数据结构中的根结点被删除后，它的某一个后件可能就变成了根结点；在一个数据结构中的终端结点后插入一个新结点后，则原来的那个终端结点就不再是终端结点而成为内部结点了。有关数据结构的基本运算将在后面讲到具体数据结构时再介绍。

如果在一个数据结构中一个数据元素都没有，则称该数据结构为空的数据结构。在一个空的数据结构中插入一个新的元素后就变为非空；在只有一个数据元素的数据结构中，将该元素删除后就变为空的数据结构。

根据数据结构中各数据元素之间前后件关系的复杂程度，一般将数据结构分为两大类型：线性结构与非线性结构。

如果一个非空的数据结构满足下列两个条件：

（1）有且只有一个根结点。

（2）每一个结点最多有一个前件，最多有一个后件。

则称该数据结构为线性结构。线性结构又称线性表。

由此可以看出，在线性结构中，各数据元素之间的前后件关系是很简单的。如例 2.1 中的一年四季这个数据结构，以及例 2.3 中的 n 维向量数据结构，它们都属于线性结构。

特别需要说明的是，在一个线性结构中插入或删除任何一个结点后还应是线性结构。根据这一点，如果一个数据结构满足上述两个条件，但当在此数据结构中插入或删除任何一个结点后就不满足这两个条件了，则该数据结构不能称为线性结构。例如，图 2.4 所示的数据结构显然是满足上述两个条件的，但它不属于线性结构这个类型，因为如果在这个数据结构中删除结点 A 后，就不满足上述条件(1)。

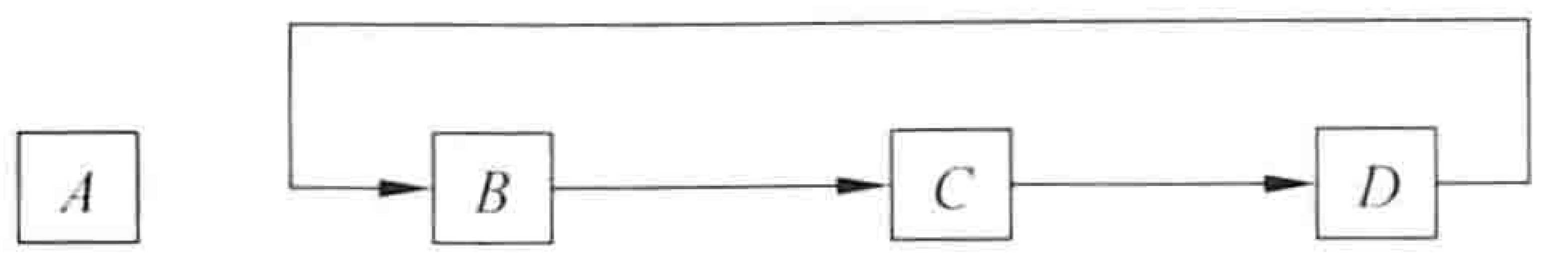


图 2.4 不是线性结构的数据结构特例

如果一个数据结构不是线性结构，则称之为非线性结构。如例 2.2 中反映家庭成员间辈分关系的数据结构，以及例 2.4 中的数据结构，它们都不是线性结构，而是属于非线性结构。显然，在非线性结构中，各数据元素之间的前后件关系要比线性结构复杂，因此，对非线性结构的存储与处理比线性结构复杂得多。

线性结构与非线性结构都可以是空的数据结构。一个空的数据结构究竟是属于线性结构还是属于非线性结构，这要根据具体情况来确定。如果对该数据结构的运算是按线性结构的规则来处理的，则属于线性结构；否则属于非线性结构。

2.2 线性表及其顺序存储结构

2.2.1 线性表及其运算

1. 什么是线性表

线性表(Linear List)是最简单、最常用的一种数据结构。

线性表由一组数据元素构成。数据元素的含义很广泛，在不同的具体情况下，它可以有不同的含义。例如，一个 n 维向量$(x_1, x_2, \cdots, x_n)$是一个长度为 n 的线性表，其中的每一个分量就是一个数据元素。又如，英文小写字母表$(a, b, \cdots, z)$是一个长度为 26 的线性表，其中的每一个小写字母就是一个数据元素。再如，一年中的四个季节(春，夏，秋，冬)是一个长度为 4 的线性表，其中的每一个季节名就是一个数据元素。

矩阵也是一个线性表，只不过它是一个比较复杂的线性表。在矩阵中，既可以把每一行看成是一个数据元素(即一个行向量为一个数据元素)，也可以把每一列看成是一个数据元素(即一个列向量为一个数据元素)。其中，每一个数据元素(一个行向量或一个列向量)实际上又是一个简单的线性表。

数据元素可以是简单项(如上述例子中的数、字母、季节名等)。在稍微复杂的线性表中，一个数据元素还可以由若干个数据项组成。例如，某班的学生情况登记表是一个复杂的线性表，表中每一个学生的情况就组成了线性表中的每一个元素，每一个数据元素包括姓名、学号、性别、年龄和健康状况 5 个数据项，如表 2.1 所示。在这种复杂的线性表中，由若

表 2.1 学生情况登记表

姓名	学号	性别	年龄	健康状况
王　强	800356	男	19	良好
刘建平	800357	男	20	一般
赵　军	800361	女	19	良好
葛文华	800367	男	21	较差
⋮	⋮	⋮	⋮	⋮

干数据项组成的数据元素称为记录(record),由多个记录构成的线性表称为文件(file)。因此,上述学生情况登记表就是一个文件,其中每一个学生的情况就是一个记录。

综上所述,线性表是由 $n(n\geqslant 0)$ 个数据元素 $a_1,a_2,\cdots,a_n$ 组成的一个有限序列。表中的每一个数据元素,除了第一个外,有且只有一个前件;除了最后一个外,有且只有一个后件。即线性表或是一个空表,或可以表示为

$$(a_1,a_2,\cdots,a_i,\cdots,a_n)$$

其中 $a_i(i=1,2,\cdots,n)$ 是属于数据对象的元素,通常也称其为线性表中的一个结点。

显然,线性表是一种线性结构。数据元素在线性表中的位置只取决于它们自己的序号,即数据元素之间的相对位置是线性的。

非空线性表有如下一些结构特征:

(1) 有且只有一个根结点 a_1,它无前件。

(2) 有且只有一个终端结点 a_n,它无后件。

(3) 除根结点与终端结点外,其他所有结点有且只有一个前件,也有且只有一个后件。线性表中结点的个数 n 称为线性表的长度。当 $n=0$ 时,称为空表。

2. 线性表的顺序存储结构

在计算机中存放线性表,一种最简单的方法是顺序存储,也称为顺序分配。

线性表的顺序存储结构具有以下两个基本特点:

(1) 线性表中所有元素所占的存储空间是连续的。

(2) 线性表中各数据元素在存储空间中是按逻辑顺序依次存放的。

由此可以看出,在线性表的顺序存储结构中,其前后件两个元素在存储空间中是紧邻的,且前件元素一定存储在后件元素的前面。

在线性表的顺序存储结构中,如果线性表中各数据元素所占的存储空间(字节数)相等,则要在该线性表中查找某一个元素是很方便的。

假设线性表中的第一个数据元素的存储地址(指第一个字节的地址,即首地址)为 $\mathrm{ADR}(a_1)$,每一个数据元素占 k 个字节,则线性表中第 i 个元素 a_i 在计算机存储空间中的存储地址为

$$\mathrm{ADR}(a_i)=\mathrm{ADR}(a_1)+(i-1)k$$

即在顺序存储结构中,线性表中每一个数据元素在计算机存储空间中的存储地址由该元素在线性表中的位置序号唯一确定。一般来说,长度为 n 的线性表

$$(a_1,a_2,\cdots,a_i,\cdots,a_n)$$

在计算机中的顺序存储结构如图 2.5 所示。

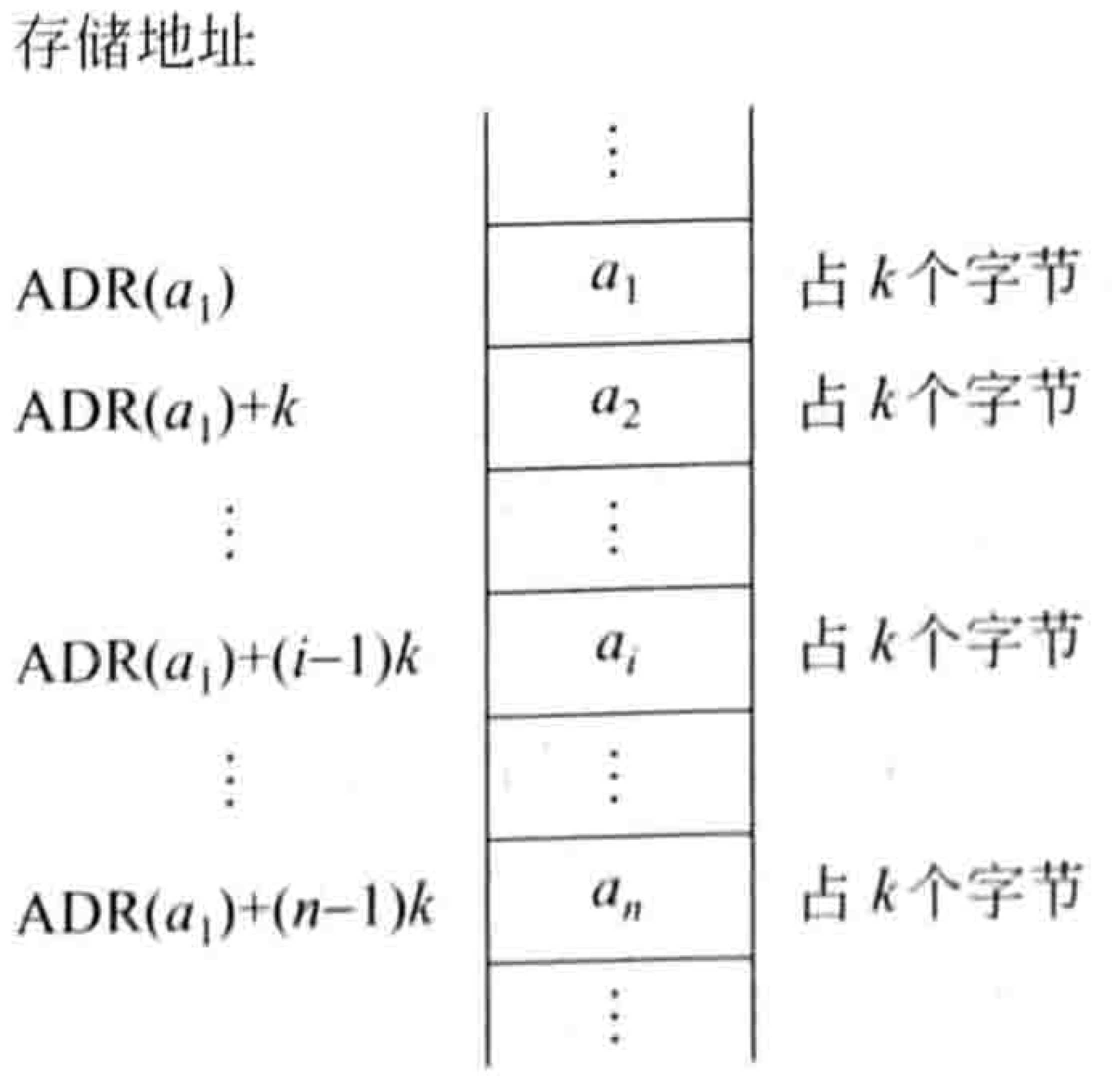


图 2.5 线性表的顺序存储结构

在程序设计语言中,通常定义一个一维数组来表示线性表的顺序存储空间。因为程序设计语言中的一维数组与计算机中实际的存储空间结构是类似的,这就便于用程序设计语言对线性表进行各种运算处理。

在用一维数组存放线性表时,该一维数组的长度通常要定义得比线性表的实际长度大一些,以便对线性表进行各种运算,特别是插入运算。在一般情况下,

如果线性表的长度在处理过程中是动态变化的，则在开辟线性表的存储空间时要考虑到线性表在动态变化过程中可能达到的最大长度。如果开始时所开辟的存储空间太小，则在线性表动态增长时可能会出现存储空间不够而无法再插入新的元素；但如果开始时所开辟的存储空间太大，而实际上又用不着那么大的存储空间，则会造成存储空间的浪费。在实际应用中，可以根据线性表动态变化过程中的一般规模来决定开辟的存储空间量。

建立一个容量为 m 的空线性表的顺序存储空间(即初始化线性表的顺序存储空间)的 C++ 描述如下：

```
//ch2_1.cpp
    template<typename T>         //模板声明,T 为类型参数
    void  init_sq_LList(T *v, int m, int *n)
    {  v=new T[m];               //动态申请存储空间
       *n=0;                     //线性表长度置 0
      return;
    }
```

在上述描述中，T 为线性表中元素的虚拟数据类型。

要释放线性表的顺序存储空间时，可以用“delete[] v;”。

在线性表的顺序存储结构下，可以对线性表进行各种处理。主要的运算有以下几种：

(1) 在线性表的指定位置处加入一个新的元素(即线性表的插入)。

(2) 在线性表中删除指定的元素(即线性表的删除)。

(3) 在线性表中查找某个(或某些)特定的元素(即线性表的查找)。

(4) 对线性表中的元素进行整序(即线性表的排序)。

(5) 按要求将一个线性表分解成多个线性表(即线性表的分解)。

(6) 按要求将多个线性表合并成一个线性表(即线性表的合并)。

(7) 复制一个线性表(即线性表的复制)。

(8) 逆转一个线性表(即线性表的逆转)。

本节先讨论线性表在顺序存储结构下的插入与删除的问题。对于线性表在其他存储方式下的插入与删除以及其他运算将放在后面的有关章节中讨论，或者作为练习由读者自己完成。

3. 线性表在顺序存储下的插入运算

首先举一个例子来说明如何在顺序存储结构的线性表中插入一个新元素。

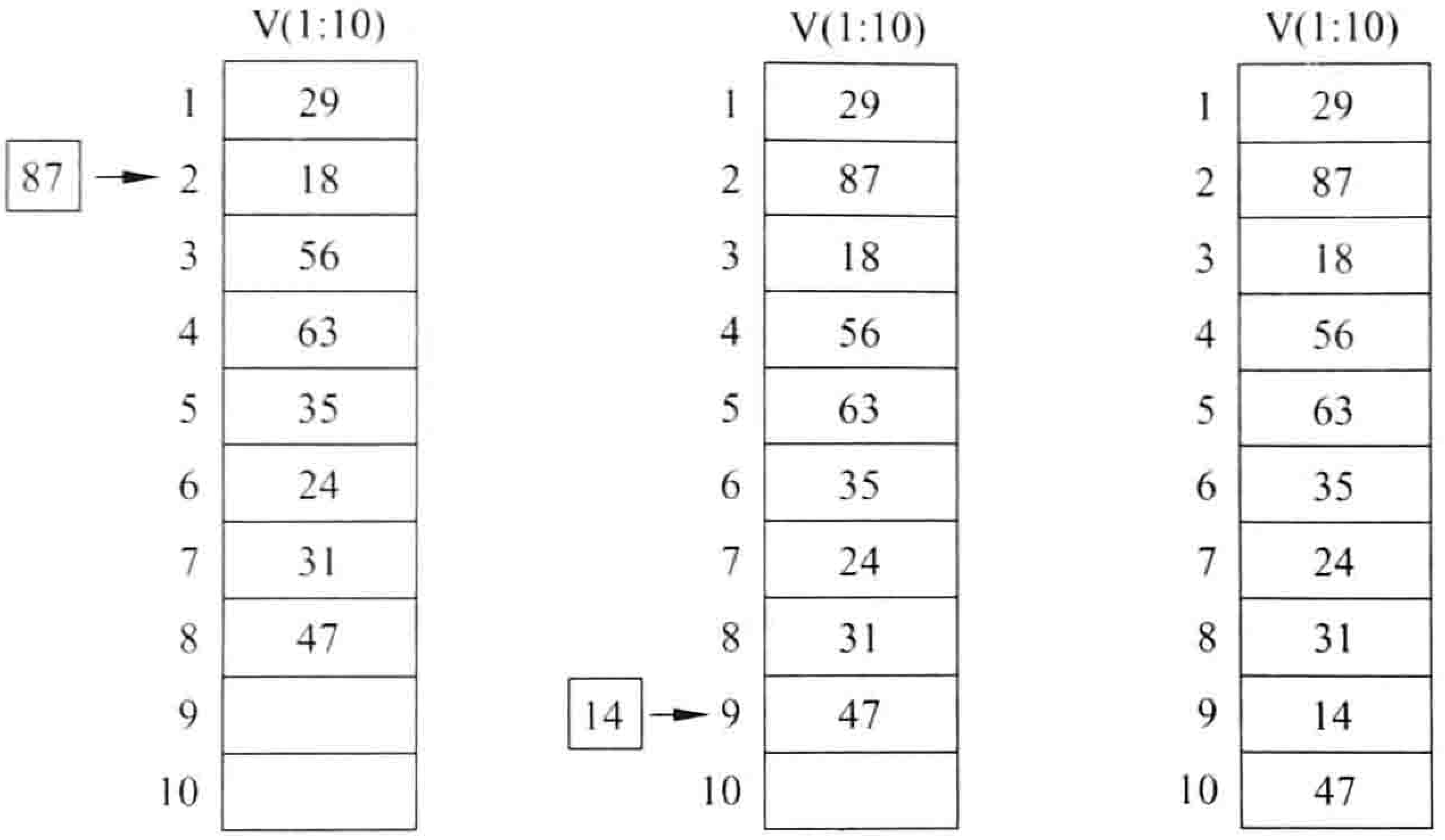
例 2.5 图 2.6(a)为一个长度为 8 的线性表顺序存储在长度为 10 的存储空间中。现在要求在第 2 个元素(即 18)之前插入一个新元素 87。其插入过程如下：

首先从最后一个元素开始直到第 2 个元素，将其中的每一个元素均依次往后移动一个位置；然后将新元素 87 插入到第 2 个位置。插入一个新元素后，线性表的长度变成了 9，如图 2.6(b)所示。

如果再要在线性表的第 9 个元素之前插入一个新元素 14，则采用类似的方法：将第 9 个元素往后移动一个位置，然后将新元素插入到第 9 个位置。插入后，线性表的长度变成了 10，如图 2.6(c)所示。

现在，为线性表开辟的存储空间已经满了，不能再插入新的元素了。如果再要插入，则

会造成称为“上溢”的错误。



(a) 长度为 8 的线性表　　(b) 插入元素 87 后　　(c) 又插入元素 14 后

图 2.6　线性表在顺序存储结构下的插入例

一般来说，设长度为 n 的线性表为

$$(a_1,a_2,\cdots,a_i,\cdots,a_n)$$

现要在线性表的第 i 个元素 a_i 之前插入一个新元素 b，插入后得到长度为 $n+1$ 的线性表为

$$(a'_1,a'_2,\cdots,a'_j,a'_{j+1},\cdots,a'_n,a'_{n+1})$$

则插入前后的两线性表中的元素满足如下关系：

$$a'_j=\begin{cases}a_j & 1\leqslant j\leqslant i-1\\ b & j=i\\ a_{j-1} & i+1\leqslant j\leqslant n+1\end{cases}$$

在一般情况下，要在第 $i(1\leqslant i\leqslant n)$ 个元素之前插入一个新元素时，首先要从最后一个（即第 n 个）元素开始，直到第 i 个元素之间共 $n-i+1$ 个元素依次向后移动一个位置。移动结束后，第 i 个位置就被空出，然后将新元素插入到第 i 项。插入结束后，线性表的长度就增加了 1。

显然，线性表采用顺序存储结构时，如果插入运算在线性表的末尾进行，即在第 n 个元素之后（可以认为是在第 $n+1$ 个元素之前）插入新元素，则只要在表的末尾增加一个元素即可，不需要移动表中的元素；如果要在线性表的第 1 个元素之前插入一个新元素，则需要移动表中所有的元素。在一般情况下，如果插入运算在第 $i(1\leqslant i\leqslant n)$ 个元素之前进行，则原来第 i 个元素之后（包括第 i 个元素）的所有元素都必须移动。在平均情况下，要在线性表中插入一个新元素，需要移动表中一半的元素。因此，在线性表顺序存储的情况下，要插入一个新元素，其效率是很低的，特别是在线性表比较大的情况下更为突出，这是因为数据元素的移动消耗了较多的处理时间。

假设线性表的存储空间为 $V(1:m)$，线性表的长度为 $n(n\leqslant m)$，插入的位置为 i（i 表示在第 i 个元素之前插入）。要插入新元素 b，则插入的过程如下：

(1) 首先处理以下三种异常情况：

- 当存储空间已满（即 $n=m$）时为“上溢”错误，不能进行插入，算法结束；
- 当 $i>n$ 时，认为在最后一个元素之后（即第 $n+1$ 个元素之前）插入；

• 当 $i<1$ 时，认为在第 1 个元素之前插入。

（2）然后从最后一个元素开始，直到第 i 个元素，其中每一个元素均往后移动一个位置。

（3）最后将新元素 b 插入到第 i 个位置，并且将线性表的长度增加 1。

线性表在顺序存储结构下插入算法的 C++ 描述如下：

```
//ch2_2.cpp
    #include<iostream>
    using namespace std;
    template<typename T>          //模板声明,T 为类型参数
    void ins_sq_LList(T *v, int m, int *n, int i, T b)
    {
        int k;
        if (*n==m)                //存储空间已满,上溢错误
         { cout<<"overflow!\n";   return; }
        if (i>*n) i=*n+1;         //默认为在最后一个元素之后插入
        if (i<1) i=1;             //默认为在第一个元素之前插入
        for (k=*n; k>=i; k--)
          v[k]=v[k-1];            //从最后一个元素开始,直到第 i 个元素均往后移动一个位置
        v[i-1]=b;                 //插入新元素
        *n=*n+1;                  //线性表长度增加 1
        return;
    }
```

由于 C++ 中数组的下标是从 0 开始的，即 C++ 数组中的第 1 个元素的下标为 0。因此，在 C++ 描述的算法中，涉及数组下标的变量均要减去 1，在本书以后的用 C++ 描述的所有算法中均是如此。

4. 线性表在顺序存储下的删除运算

与线性表的插入运算一样，首先举一个例子来说明如何在顺序存储结构的线性表中删除一个元素。

例 2.6 图 2.7(a)为一个长度为 8 的线性表顺序存储在长度为 10 的存储空间中。现在要求删除线性表中的第 1 个元素（即删除元素 29）。其删除过程如下：

从第 2 个元素开始直到最后一个元素，将其中的每一个元素均依次往前移动一个位置。此时，线性表的长度变成了 7，如图 2.7(b)所示。

如果再要删除线性表中的第 6 个元素，则采用类似的方法：将第 7 个元素往前移动一个位置。此时，线性表的长度变成了 6，如图 2.7(c)所示。

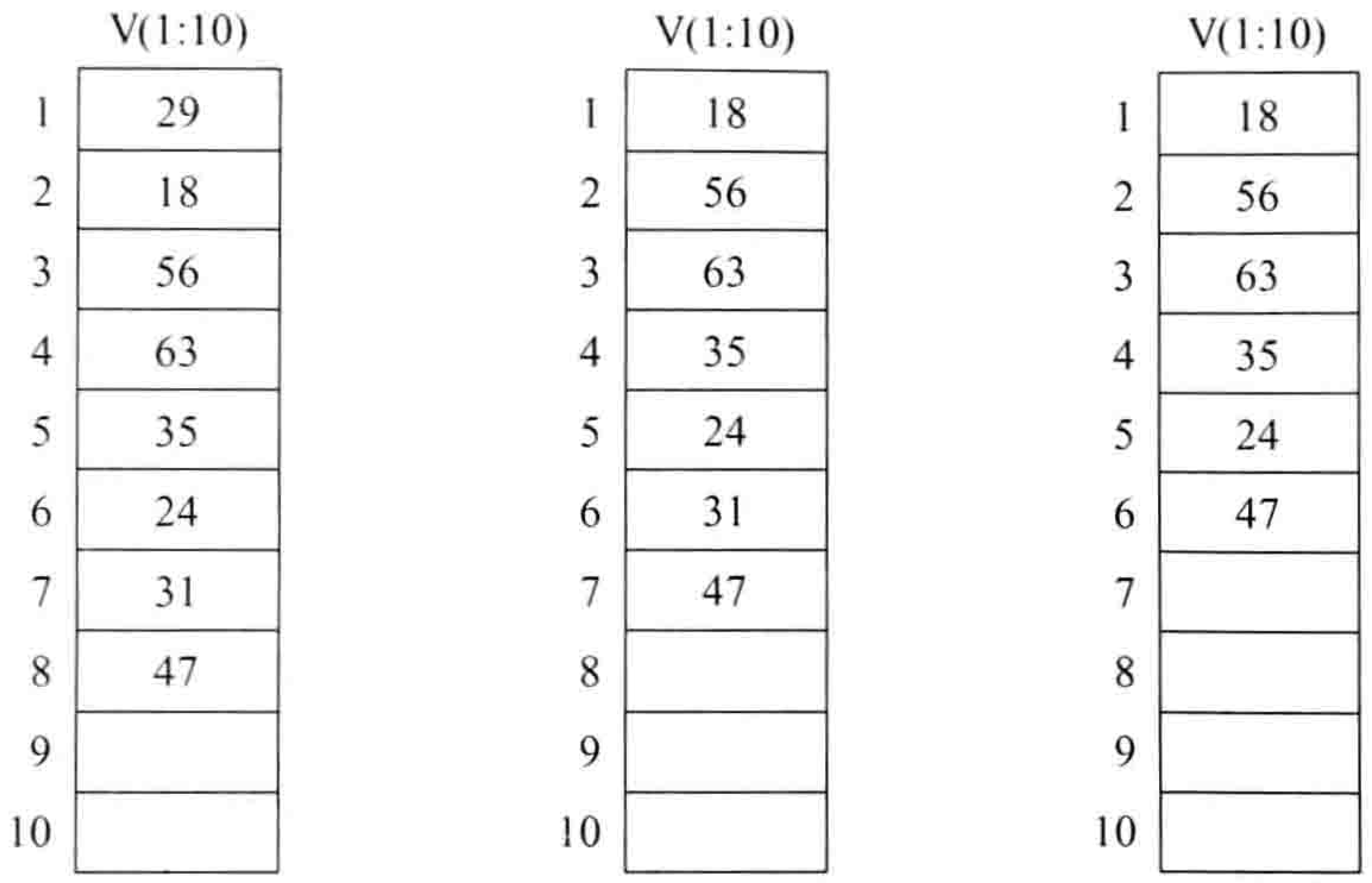
一般来说，设长度为 n 的线性表为

$$(a_1, a_2, \cdots, a_i, \cdots, a_n)$$

现要删除第 i 个元素，删除后得到长度为 $n-1$ 的线性表为

$$(a'_1, a'_2, \cdots, a'_j, \cdots, a'_{n-1})$$

则删除前后的两线性表中的元素满足如下关系：



(a) 长度为 8 的线性表　　(b) 删除元素 29 后　　(c) 又删除元素 31 后

图 2.7　线性表在顺序存储结构下的删除例

$$a'_j=\begin{cases}a_j & 1\leqslant j\leqslant i-1\\ a_{j+1} & i\leqslant j\leqslant n-1\end{cases}$$

在一般情况下，要删除第 $i(1\leqslant i\leqslant n)$ 个元素时，则要从第 $i+1$ 个元素开始，直到第 n 个元素之间共 $n-i$ 个元素依次向前移动一个位置。删除结束后，线性表的长度就减小了 1。

显然，在线性表采用顺序存储结构时，如果删除运算在线性表的末尾进行，即删除第 n 个元素，则不需要移动表中的元素；如果要删除线性表中的第 1 个元素，则需要移动表中所有的元素。在一般情况下，如果要删除第 $i(1\leqslant i\leqslant n)$ 个元素，则原来第 i 个元素之后的所有元素都必须依次往前移动一个位置。在平均情况下，要在线性表中删除一个元素，需要移动表中一半的元素。因此，在线性表顺序存储的情况下，要删除一个元素，其效率也是很低的，特别是在线性表比较大的情况下更为突出，由于数据元素的移动而消耗较多的处理时间。

假设线性表的存储空间为 $V(1:m)$，线性表的长度为 $n(n\leqslant m)$，删除的位置为 i（表示删除第 i 个元素），则删除的过程如下：

（1）首先处理以下两种异常情况：

- 当线性表为空（即 $n=0$）时为“上溢”错误，不能进行插入，算法结束；
- 当 $i<1$ 或 $i>n$ 时，认为在最后一个元素之后（即第 $n+1$ 个元素之前）插入。

（2）然后从第 $i+1$ 个元素开始，直到最后一个元素，其中每一个元素均依次往前移动一个位置。

（3）最后将线性表的长度减 1。

线性表在顺序存储结构下删除算法的 C++ 描述如下：

```
//ch2_3.cpp
    #include<iostream>
    using namespace std;
    template<class T>          //模板声明,T为类型参数
    void del_sq_LList(T *v, int m, int *n, int i)
```

```
{
    int k;
    if (*n==0)              //线性表为空,下溢错误
      { printf("underflow!\n");  return; }
    if ((i<1)||(i>*n))  //线性表中没有这个元素
      {
        cout<<"Not this element in the list!\n";
        return;
      }
    for (k=i; k<*n; k++)
     v[k-1]=v[k];           //从第 i 个元素开始,直到最后一个元素均往前移动一个位置
    *n=*n-1;                //线性表长度减 1
    return;
}
```

从线性表在顺序存储结构下的插入与删除运算可以看出,线性表的顺序存储结构对于小线性表或者其中元素不常变动的线性表来说是合适的,因为顺序存储的结构比较简单。但这种顺序存储的方式对于元素经常需要变动的大线性表就不太合适了,因为插入与删除的效率比较低。

5. 顺序表类

从前面对顺序表的插入与删除算法描述中可以看出,对一个顺序表的插入与删除是通过调用插入函数或删除函数来实现的,其中插入函数与删除函数是普通函数,它们对所有的顺序表是公开的。在这种机制中,任何用户都可以通过调用插入函数或删除函数对任意一个顺序表进行插入或删除操作,这是一种面向过程的程序设计方法。C++ 既可用于面向过程的结构化程序设计,又可用于面向对象的程序设计。利用 C++ 支持面向对象的机制,通过类结构可将顺序表这种数据结构和一些常用的基本运算封装在一起。

下面的描述是将顺序表的数据和基本操作(初始化、输出、插入与删除操作)封装成一个 sq_LList 类:

```
//sq_LList.h
    #include<iostream>
    using namespace std;
    template<class T>               //模板声明,数据元素虚拟类型为 T
    class  sq_LList                 //顺序表类
    { private:                      //数据成员
        int mm;                     //存储空间容量
        int nn;                     //顺序表长度
        T *v;                       //顺序表存储空间首地址
      public:                       //成员函数
        sq_LList(){ mm=0; nn=0; return;}
        sq_LList(int);              //建立空顺序表,申请存储空间
        void prt_sq_LList();        //顺序输出顺序表中的元素与顺序表长度
        int flag_sq_LList();        //检测顺序表的状态
        void ins_sq_LList(int, T);  //在表的指定元素前插入新元素
```

```
    void del_sq_LList(int);        //在表中删除指定元素
};
//建立空顺序表
template<class T>
sq_LList<T>::sq_LList(int m)
{ mm=m;                           //存储空间容量
  v=new T[mm];                    //动态申请存储空间
  nn=0;                           //顺序表长度为 0,即建立空顺序表
  return;
}
//顺序输出顺序表中的元素与顺序表长度
template<class T>
void sq_LList<T>::prt_sq_LList()
{ int i;
  cout<<"nn="<<nn<<endl;
  for (i=0; i<nn; i++) cout<<v[i]<<endl;
  return;
}
//检测顺序表的状态
template<class T>
int sq_LList<T>::flag_sq_LList()
{ if (nn==mn)  return(-1);        //存储空间已满,返回-1
  if (nn==0)  return(0);          //顺序表为空,返回 0
  return(1);                      //正常返回 1
}
//在表的指定元素前插入新元素
template<class T>
void sq_LList<T>::ins_sq_LList(int i, T b)
{ int k;
  if (nn==mm)                     //存储空间已满,上溢错误
     { cout<<"overflow"<<endl;  return; }
  if (i>nn) i=nn+1;               //默认为在最后一个元素之后插入
  if (i<1) i=1;                   //默认为在第一个元素之前插入
  for (k=nn; k>=i; k--)
    v[k]=v[k-1];                  //从最后一个元素直到第 i 个元素均后移一个位置
  v[i-1]=b;                       //插入新元素
  nn=nn+1;                        //顺序表长度加 1
  return;
}
//在顺序表中删除指定元素
template<class T>
void sq_LList<T>::del_sq_LList(int i)
{ int k;
  if (nn==0)                      //顺序表为空,下溢错误
      { cout<<"underflow!"<<endl;  return; }
```

```
    if ((i<1)||(i>nn))              //顺序表中没有这个元素
        { cout<<"Not this element in the list!"<<endl;
          return;
        }
    for (k=i; k<nn; k++)
      v[k-1]=v[k];                  //从第 i 个元素直到最后一个元素均前移一个位置
    nn=nn-1;                        //顺序表长度减 1
    return;
}
```

下面是使用顺序表类的主函数的例子。

例 2.7 建立容量为 100 的空顺序表,然后输出该空顺序表。在该顺序表中依次在第 0 个元素前插入 1.5、在第 1 个元素前插入 2.5 以及在第 4 个元素前插入 3.5,再输出该顺序表。依次删除该顺序表中的第 0 个元素以及删除该顺序表中的第 1 个元素,再输出该顺序表。主函数如下:

```
//ch2_4.cpp
    #include "sq_LList.h"
    int main()
    { sq_LList<double> s1(100);     //建立容量为 100 的空顺序表对象 s1
      cout<<"第 1 次输出顺序表对象 s1:"<<endl;
      s1.prt_sq_LList();
      s1.ins_sq_LList(0,1.5);       //在第 0 个元素前插入 1.5
      s1.ins_sq_LList(1,2.5);       //在第 1 个元素前插入 2.5
      s1.ins_sq_LList(4,3.5);       //在第 4 个元素前插入 3.5
      cout<<"第 2 次输出顺序表对象 s1:"<<endl;
      s1.prt_sq_LList();
      s1.del_sq_LList(0);           //删除顺序表 s1 中的第 0 个元素
      s1.del_sq_LList(2);           //删除顺序表 s1 中的第 1 个元素
      cout<<"第 3 次输出顺序表对象 s1:"<<endl;
      s1.prt_sq_LList();
      return 0;
    }
```

上述程序的运行结果如下:

```
第 1 次输出顺序表对象 s1:
nn=0
第 2 次输出顺序表对象 s1:
nn=3
2.5
1.5
3.5
Not this element in the list!       //指删除顺序表 s1 中的第 0 个元素
第 3 次输出顺序表对象 s1:
nn=2
```

／# 有关此电子书的说明

本人可以帮助你找到你要的PDF电子书，计算机类，文学，艺术，设计，医学，理学，经济，金融等等。质量都很清晰，为方便读者阅读观看，每本100%都带可跳转的书签索引和目录，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 QQ1779903665。

PDF代找说明：
本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ，大部分我都可以找到，而且每本100%带书签索引目录。因PDF电子书都有版权，请不要随意传播，如果您有经济购买能力，请尽量购买正版。

提供各种书籍的pd电子版代找服务，如果你找不到自己想要的书的pdf电子版，我们可以帮您找到，如有需要，请联系 QQ 1779903665.

备用:QQ 461573687
若以上联系方式失效，您可通过以下电子邮件获取有效联系方式。
E-mail：ebooksprite@163.com
E-mail：ebooksprite@foxmail.com

若您没有QQ通讯工具，请发送您的请求到 ebooksprite@gmail.com 与客服取得联系。

声明：本人只提供代找服务，每本100%索引书签和目录，因寻找和后期制作pdf电子书有一定难度，仅收取代找费用。如因PDF产生的版权纠纷，与本人无关，我们仅仅只是帮助你寻找到你要的pdf而已。